An Evaluation of Different DLP Alternatives for the Embedded Media Domain

Esther Salamí, Jesús Corbal, Mateo Valero *
Departament d'Arquitectura de Computadors,
UPC. Universitat Politècnica de Catalunya–Barcelona, Spain
e-mail: {esalami,jcorbal,mateo}@ac.upc.es

Roger Espasa
Compaq Computer Corp.,
Shrewsbury, MA
e-mail: espasa@vssad.hlo.dec.com

Abstract

The importance of media processing has produced a revolution in the design of embedded processors. In order to face the high computational and technological demands of near future media applications, new embedded processors are including features that were commonly restricted to the general purpose and the supercomputing domains. In this paper we have evaluated the performance of various DLP (Data Level Parallelism) oriented embedded architectures and analyzed quantitative data in order to determine the highlights and disadvantages of each approach. Additionally we have analyzed the differences between the explicit parallel versions of code (often based on the standard algorithms) and the high-tuned, non-vectorizable versions usually found in real multimedia programs. We will show that sub-word SIMD architectures (like MMX) are a very cost-effective solution, and that, while long vector architectures provide few improvements at a very high cost, a smart combination between vector and SIMD-like architectures is the alternative that leverages best performance at a reasonable cost. We will also show that the memory latency tolerance, typical of vector architectures, is partially compensated by the worse spatial locality found when executing vector code.

1 Introduction

The significance that media processing has been taking on during the last years have not been limited to the general purpose domain. On the contrary, the embedded domain has experimented a revolution based on new and harder demands. Near future applications such as *personal mobile computing*, WebTV devices, DVD players or even next generation of game consoles [1] are just a few examples.

*This work has been supported by Direccio General de Recerca de la Generalitat under grant 1998FI-00260, by the Ministry of Education of Spain under contract CICYT TIC98-05110C02-01 and by the CEPBA.

The 32-bit embedded processors have already narrowed the gap between embedded and desktop systems [2] and DSP processors include currently features that were restricted not far ago to just the general purpose domain.

Realizing the computational demands, together with the cost and power consumption requirements of these new applications, it can be easily predicted that even more aggressive approaches are going to be implemented in future embedded processors.

In this paper we will try to evaluate several of the various DLP oriented approaches designed to boost the performance of multimedia and DSP applications for the embedded domain: (a) sub-word level SIMD multimedia architectures, (b) conventional short/long vector architectures and (c) matrix SIMD multimedia architectures. In order to understand the performance benefits of each alternative we will present quantitative data such as the number and type of instructions executed, the overall number of operations or the memory behavior.

As a side matter of study, this paper focuses also on studying the difference between scalar optimized code and explicit DLP code, which is a generally overlooked issue in most multimedia papers. We will show what are the main optimizations made over the standard vectorizable algorithms and we will analyze both optimized and vectorizable versions.

2 The embedded domain evolution towards media processing

Media processing has motivated strong changes in the focus and design of mid 90s processors. In the general purpose domain, these changes have been very straightforward with the inclusion of SIMD-like multimedia extensions such as *MMX* [3], *VIS* [4] or *MDMX* [5]. These extensions have become the most important change to the basic ISA since the inclusion of the FP units inside the processor core.

On the other hand, the changes in the embedded design have been strongly influenced by different domains such as the general purpose or the supercomputing domain.

From the general purpose domain, two main DSP architectures have evolved: superscalar DSPs with SIMD instructions (such as TriCore [6]) and VLIW DSPs, either with special SIMD instructions (Philips TriMedia [7], TigerSHARC [8]) or without (Texas Instruments TMS320C6201 [9]).

From the supercomputing domain, the vector and systolic paradigms have influenced new DSP processors. There have been several papers dealing with the design of cost-effective vector microprocessors [10, 11]. Examples of vector microprocessor designs are the Torrent T0 [12] or the V-IRAM project [13]. Additionally, there are current projects using streaming SIMD architectures to address 3D graphics processing such as Imagine [14]. Finally, as a side research line, there have been papers dealing with the inclusion to a basic superscalar core of a conventional vector ISA extension [15] and a matrix ISA extension [16].

While all these new architectures are able to take benefit from the abundant *Data Level Parallelism* available in common media codes, there has not been any work comparing their potential performance and characteristics, specially in the embedded domain. When dealing with restricted resources, different factors such as the code density, the explicit parallelism or the memory behavior can affect performance in different ways than more aggressive general purpose architectures. These three factors will be the basis of the performance analysis in this paper.

3 Evaluation Background

In this section we will discuss the selection of benchmarks for our set of evaluations, the impact of the scalar optimized code compared to explicitly parallel codes, and we will briefly describe all the modeled architectures and the approach used for simulate them.

3.1 Benchmarks

The difficulty to capture all of the essential elements of modern embedded multimedia and communications systems is reflected on the lack of any standardized benchmark suite. For our study we focus on three representative programs of image and video workloads: `cjpeg` (a JPEG image encoder), `djpeg` (a JPEG image decoder) and `mpeg2encode` (a MPEG2 video encoder), all from the UCLA *Mediabench* suite [17].

We have chosen entire applications instead of representative kernels in order to compare the real effectiveness of current architectures. Note that the improvement obtained in some parts of the code can be overridden by the degradation produced on the remaining part. The selected programs are representative enough as far as compilation methods and vectorization percentage is concerned.

JPEG is a compression standard for either grayscale and color digital images based on the DCT-method [18]. The codification is performed in three stages: *color space conversion* and *downsample*,*forward DCT transform and quantization*, and *entropy coding*. In color space conversion, each pixel from the source image in converted from the RGB to its YUV representation and then the chrominance components (U and V) are downsampled by a factor of two on both spatial dimensions. The forward DCT processing step lays the foundation for achieving data compression by concentrating most of the signal in the lower spatial frequencies. Source images samples are grouped into 8x8 blocks and input to the DCT. The output is another block of 64 coefficients with the property that most of them have zero or near-zero amplitude and need not be encoded. Afterwards, each coefficient is quantized with the purpose to achieve further compression by representing the coefficients with no greater precision than is necessary to achieve the desired image quality. Finally, all the quantized coefficients are ordered into a "zig-zag" sequence, so that they can be encoded more compactly based on their statistical characteristics (Huffmann coding). `Djpeg` just performs the inverse operations in the reverse order.

MPEG-2 video compression standard was developed by the *Motion Picture Experts Group* [19]. Video sequences usually contain statistical redundancies in both temporal and spatial direction. Spatial correlation is exploited for each frame in the same way as JPEG, and motion compensated prediction techniques are used to reduce temporal redundancies between frames. *Motion estimation* searches which block of the previous image matches better with the block being compressed (this becomes the most computational-intensive part of the process), and the resulting displacement between the two blocks is called the motion vector. Usually, the block size is 16x16 pixels for the luminance component (Y) and 8x8 for the chrominance components (U and V). A motion compensated difference block is then formed by subtracting the pixel values of the predicted block from that of the current block. The different block is then transformed, quantized and entropy coded.

3.2 Explicit DLP Vs Optimized Scalar Code

Most of the algorithms used in the standards above have a vector nature. Nevertheless, due to the intrinsic significance of most multimedia algorithms, there has been a great effort focusing on reducing the overall number of required operations. Unfortunately, this effort has been oriented towards scalar architectures, hiding in most cases the data parallel nature of the original algorithm.

We can find the most representative example in the DCT algorithm. This transformation can be represented as a matrix operation using a 8x8 transform matrix A to obtain the 8x8 transform coefficients matrix C based on a bilinear transformation: $C = A \cdot B \cdot A^T$, where B is the input block and A^T denotes the transpose of A. This would involve 1024 multiplications for each input block. Nevertheless, various fast algorithms have been introduced in the literature for reducing the number of multiplications involved in the transform [20]. The algorithm used in the JPEG standard only needs to perform 192 products to produce one resultant block; but because of this optimization, the new code cannot be vectorized.

The use of memory tables to replace multiplications or other operations groups is quite frequent too. In color space conversion, the equations to be implemented for each pixel are:

$$Y = C1 * R + C2 * G + C3 * B$$
$$U = C4 * R + C5 * G + C6 * B + 128$$
$$V = C6 * R + C7 * G + C8 * B + 128$$

where $C1$ to $C8$ are constants and R, G and B are the pixel color components. To avoid floating-point/fixed-point conversions, fractional numbers are represented as integers scaled up by 2^{16}. Moreover, in order to avoid doing multiplications in the inner loop, these products are precalculated for all possible values of R, G and B. This would involve 256 entries of 32 bits per table; as two multiplication constants are identical, only eight tables are needed. These small tables are grouped into one unique table of 8 Kbytes, which can be held in cache. Taking into account that offsets and rounding factors are included in the tables, the real change is 9 loads and 6 adds in opposite to 9 mults and 9 adds.

Something similar is done to perform saturation in `djpeg`. Several decompression processes need to range-limit values between 0 and 255. On most machines a table lookup is faster than the explicit test:

```
if(x < 0) x = 0;
else if (x > 255) x = 255;
```

So, two conditional branches and an assignment above are replaced by one single memory access.

Finally, another typical scalar optimization which can prevent a code fragment from being vectorized is a break condition inside a loop. We can look at the distance function in `mpeg2encode` for a sample. In motion estimation, the distance between two blocks is computed as the sum of absolute differences between the pixels of both blocks. In fact, the object of the search is finding the block with minimal distance, so it is not worth to calculate the full distance when the sum accumulated for some columns exceeds the current minimum, and the resulting code is something similar to this:

```
for(i=0;i<h;i++) {
    for(j=0;j<w;j++)
        s += abs(b1[i][j]-b2[i][j]);
    if (s >= distlim) break;
}
```

We cannot vectorize this code over loop i because of the conditional break, but this line of code could be removed without affecting the program output but rather the overall number of operations executed.

Note that the improvement obtained with this kind of optimizations is strongly architecture dependent. We have looked for the main optimizations which are present in the three benchmarks and handwritten the standard (without optimizations) associated code, so that we can generate vector/SIMD code. In section 4 we will evaluate the impact of the optimized code over performance.

The optimizations we have taken into account are:

a) For *cjpeg*: tables used for color space conversion have been changed for the lineal combination of the inputs showed above. The DCT is implemented as matrix per matrix products.

b) For *djpeg*: besides the optimizations described for *cjpeg*, saturation is performed as a comparison instead of just accessing a table. In the optimized code, they avoid computing the IDCT for those blocks whose elements are all zero. Initial evaluations showed us that eliminating this optimization would provide us diminishing returns. Therefore, the IDCT has not been modified for any vector code (except for the basic non-optimized reference code).

c) For *mpeg2encode*: as in the previous benchmarks, the IDCT is implemented as two matrix per matrix products. In motion estimation, we have removed the break condition in distance computation when generating matrix-oriented code (see subsection 3.3) since would not allow us to vectorize the whole matrix, and also when generating MMX-like code as it would not allow us to generate optimal scheduling.

3.3 Modeled Architectures

In this paper, we are going to evaluate five different SIMD/vector architectures:

- 2-way basic superscalar DSP
- 2-way superscalar DSP + SIMD extensions
- 2-way DSP + *long vector* instructions (128)
- 2-way DSP + *short vector* instructions (16)
- 2-way DSP + *Matrix* SIMD extensions

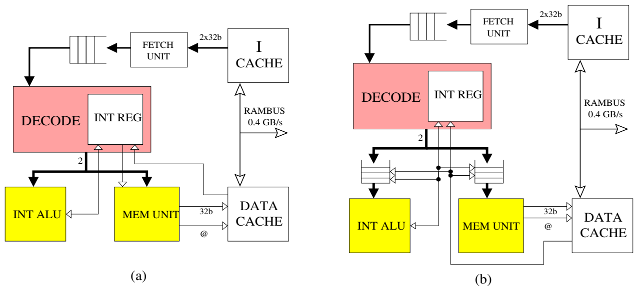

Figure 1. The basic superscalar DSP reference architecture: (a) in-order, (b) out-of-order.

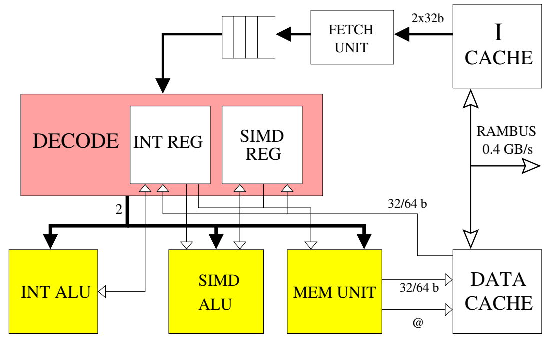

Figure 2. The in-order superscalar DSP + SIMD instructions architecture.

The basic characteristics of all the architectures (such as the issue rate, the latencies, the pipeline or the cache configurations) have been chosen based on those of the Siemens TriCore TC10. Note that it is very difficult to provide a rather perfectly fair comparison between architectures, since factors such as the characteristics of the specific ISA or the election of the number of the functional units may affect overall performance. This paper is an attempt to determine trends and characteristics rather than identifying the best alternative.

3.3.1 Reference superscalar DSP

The basic reference DSP is a 100MHz, 2-way issue, RISC load/store architecture with 32 general purpose registers. The engine is able to fetch and decode up to two different instructions per cycle. There are two different pipelines: one for integer and branch instructions and one for memory instructions. The integer operation latencies are 1 single cycle, except for multiply operations (3 cycles) and divide operations (7 cycles).

The processor has a 16KB, 2-way set associative instruction cache that provides 16-byte cache lines into an instruction buffer that decouples the fetch and the decode stages. The data cache is a 16KB, write-back, 4-way set associative cache with 32-byte cache lines able to provide one 32-bit data access per cycle. The processor is coupled to a pseudo *Direct Rambus* memory system with a bi-directional, 128-bit wide, 25MHz main bus, able to deliver up to 0.4 GB/s.

The in-order execution version has a very simple control logic that stalls the pipeline whenever we encounter any kind of data dependence or resource constraint. The out-of-order version of the DSP (see figure 1.b) provides register renaming (40 physical registers) and includes one reservation station (of 8 slots) per pipeline. Instructions between pipelines can be executed out-of-order, but instructions inside the same reservation station must be executed in-order.

In order to evaluate the performance of the architecture we have used *ATOM* [21] to provide *Alpha* code traces to our Jinks simulator [15]. In order to approximate the effect of 0-cycle resolution branches typical of several DSPs, we have taken an optimistic approach and have assumed perfect branch prediction.

3.3.2 Superscalar DSP + SIMD extensions

We have enhanced the basic model with a SIMD register file together with an additional pipeline able to execute 64-bit sub-word level SIMD instructions (see figure 2). We have used the emulation libraries described in [16] to hand-write the same applications with a SIMD ISA extension fairly similar to INTEL's SSE [22] integer opcodes. This SIMD extension provides 67 opcodes and 32 64-bit SIMD registers able to operate on up to eight 8-bit items in parallel. The data path of the data cache has been enlarged to allow one full byte-wise 64-bit access per cycle. For more information about the emulation and simulation details the reader may refer to [16, 23].

3.3.3 DSP + vector instructions

We have enhanced the basic model with a vector register file and a real single vector unit. The in-order execution model is similar to that proposed in [11], while the out-of-order version is based on the architecture proposed in [15]. Both the register file and the vector functional unit are clusterized in 4 independent 32-bit vector pipes (or lanes) where the different vector elements are interleaved (see figure 3). Therefore, up to 4 operations from the same vector instruction can be performed per cycle (in the same vein that the SSE sub-word level parallelism). We have used the CONVEX C4000 compiler to generate vector code and we have generated traces to feed *Jinks*. The CONVEX ISA provides 16 32-bit logical vector registers, and 52 address and scalar registers. The out-of-order version of the architecture performs register renaming with 24 physical vector registers

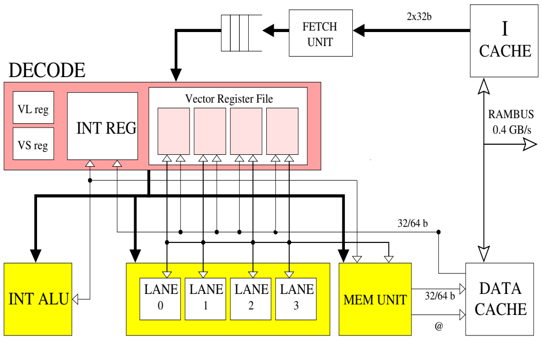

Figure 3. The in-order vector microprocessor DSP reference architecture.

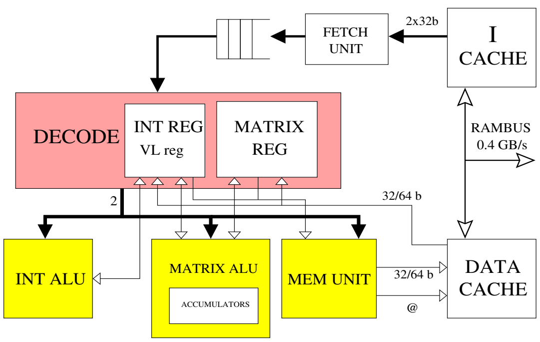

Figure 4. The in-order superscalar DSP + Matrix instructions architecture.

and 64 scalar/address registers.

In order to study the design trade-offs, we have generated code for two different vector lengths: 16 and 128. For the 16-length version, we have used strip-mining techniques on all those vectorizable loops it was required. For the full vector length version (128), whenever possible, we used loop-unrolling plus loop-interchange techniques so that we could achieve the longest effective vector length possible.

SIMD-like extensions such as MMX are able to provide up to 8 elements with a single 64-bit memory access. We assume that the vector microprocessor memory system is able to provide the same bandwidth when the vector elements are consecutively arranged in memory (up to 8 elements of 8 bits, or up to 4 elements of 16 bits) and that the logic to distribute the elements among the vector lanes does not add additional cycles of latency.

3.3.4 DSP + matrix SIMD extensions

In [16] we proposed a matrix ISA that is basically an hybrid between conventional vector ISAs and SIMD MMX-like ISAs. This ISA is able to exploit DLP from two different dimensions (parallel loops). We used emulation libraries to

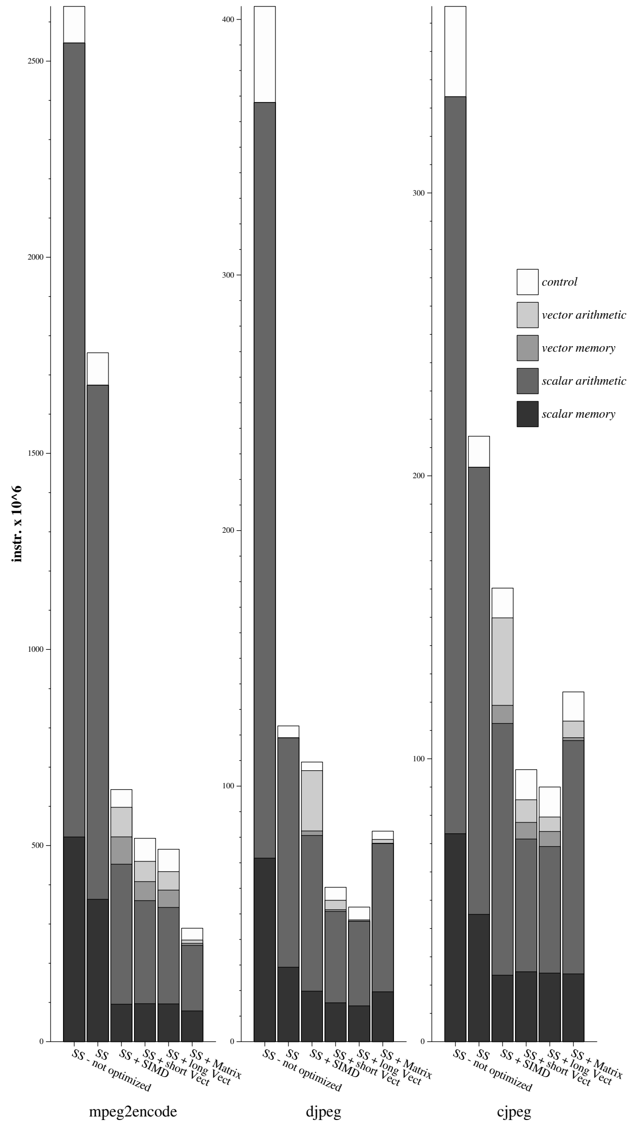

Figure 5. Dynamic instruction breakdown.

hand-write the code using the proposed model and Jinks was modified to be able to detect the emulation functions calls while gathering the SIMD traces.

We have modified the DSP+vector model in order to allow the execution of this kind of instructions. The ISA provides 16 logical matrix registers (with 16 64-bit words each) and 2 logical 192-bit packed accumulators (similar to those proposed in the MDMX multimedia extension [5]). We have an independent matrix register file and an independent accumulator file. In sharp contrast with the conventional vector version, we have not implemented parallel lanes since: a) we already have the MMX-like sub-word execution capabilities, and b) the accumulators complicate the design of fully independent lanes. The out-of-order version provides register renaming with 20 physical matrix registers and 4

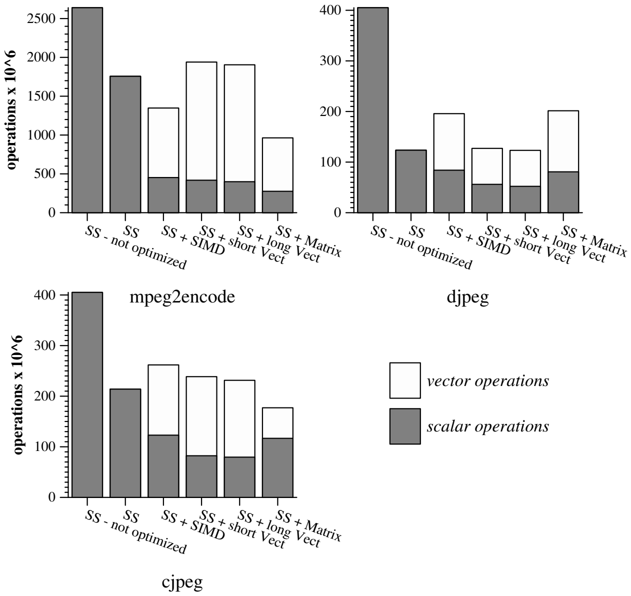

Figure 6. Dynamic operation count.

physical multimedia accumulators.

4 Quantitative Analysis

In this section we will provide quantitative data in order to identify the main characteristics of every architecture under study. We will show the instruction and operation breakdowns and an analysis of the performance results. Additionally, we will study the impact of out-of-order execution over each architecture. Finally, we will analyze the data locality and the memory latency to explain the impact of a realistic memory over performance.

4.1 Instruction Breakdown

Figure 5 shows the dynamic instruction breakdown for the benchmarks under study. The instruction types have been classified into the following five categories: scalar memory, scalar arithmetic, vector memory, vector arithmetic and branches (control).

From the results, we can clearly observe that the explicit vector parallel versions of code have a huge instruction overhead. The optimizations described in section 3.2 are able to reduce between 35% and 70% of the overall number of instructions. Note specially the high degradation introduced in the non-optimized code for `djpeg`, mainly produced by the zero condition removal mentioned in subsection 3.2, since almost 3/4 parts of the input blocks have zero AC coefficients.

Because of the lack of resources in typical embedded architectures, the increases in performance are almost pro-

	SS + SIMD	SS + SV	SS + LV	SS + Matrix
mpeg2encode	6.60	15.19	16.47	50.82
djpeg	4.71	15.99	127.66	75.10
cjpeg	3.56	11.27	14.53	11.48

Table 1. Average vector length of the instructions for all the SIMD architectures.

portional to the reduction of the number of instructions (considering the same ISA). Therefore, we may expect that the optimized version of code is going to execute about twice faster than the vectorizable version of code.

Taking into account that all the SIMD/vector versions of code are based on the non-optimized vectorizable version, it might seem that the overall number of instructions could be higher than in the optimized original version. Nevertheless, as seen in the figure, the MMX-like SIMD architecture executes about 35% fewer instructions than the optimized code, and the vector oriented architectures (short, long and matrix) an average of 60% fewer instructions. The reason is that, as already pointed out in [24], the SIMD/vector architectures can pack several scalar memory and arithmetic instructions into a single vector/SIMD instruction. Moreover, due to the fact that multiple instances of a loop are replaced with equivalent SIMD/vector instructions, there is an additional reduction of the number of instructions involved in the loop-related control (that is, loop indexes and address variables). Finally, there is another factor of reduction related to the specific characteristics of every ISA. For instance, the MMX-like and the matrix SIMD ISAs have saturation arithmetic that avoids having to perform the process described in section 3.2 and the matrix SIMD ISA allow to perform multiply&accumulate instructions.

4.2 Operation Breakdown

The overall instruction reduction is strongly dependent on three factors: the average vector length (say, the average number of operations per vector instruction), the vectorization percentage of the program and the overall number of operations to execute. Figure 6 shows the dynamic operation count and table 1 shows the average vector length for all the benchmarks under study.

Analyzing the average vector length for every architecture we can see that the short vector architecture is able to leverage a vector length very close to the maximum (16). The SIMD architecture is also able to provide convincing vector lengths (packing only 3 times less elements than the short vector architecture with a vector register file 16 times smaller). On the other hand, the average vector length of the long vector architecture is rather disappointing, since only for `djpeg` is able to almost reach its maximum vector length, and for the other benchmarks is barely able to leverage longer lengths than the short vector alternative. This is due to the fact that most multimedia kernels are characterized by having several nested loops with very small loop counts (between 8 and 16). The number of cases where longer loops or loop interchange can be used is very small. In sharp contrast with the conventional vector architectures, the matrix ISA, due to its capability to vectorize two inner loops, leverages very long vectors lengths for `mpeg2encode` and `djpeg` and a fair vector length for `cjpeg`.

The overall number of operations is a way to determine the semantic richness of every ISA and is a strong indicator of the final performance. From the results in figure 6 we can identify three different cases. In `mpeg2encode` both the MMX-like SIMD and the matrix architecture performs less operations than the conventional vector architectures. This is explained by the fact that they have very powerful instructions to perform the *motion estimation* algorithm, such as the *vector average* or the *sum of absolute differences* instructions. Furthermore, the matrix architecture only needs to execute almost half the number of operations than the SIMD architecture thanks to the advantages of the matrix accumulators [16]. Additionally, the vectorization percentage of the overall program is similar, since just this algorithm represents almost a 80% of the entire program.

Unfortunately, the similarity in the vectorization percentage is not found in the other two benchmarks. The CONVEX compiler is able to vectorize much more instructions than simply identifying some kernels and hand-write them (as is the case in the SIMD and matrix versions). This causes that the conventional vector architectures execute fewer scalar instructions than the others. Moreover, the number of vector operations is also smaller. This is produced by the large overhead involved in packing/unpacking operations (usually found in MMX-like ISAs) to perform typical transformations such as *data promotion*, *matrix transpose* or *sign conversion*. This logic overhead may represent almost the 50% of the overall number of operations. As we will see later, in some cases (such as `djpeg`) this overhead may cause less performance but, interesting enough, for some other cases (`cjpeg`) it may end up being beneficial.

4.3 Performance results

Figure 7 shows the performance results for the three benchmarks with realistic cache simulation. The speed-up performance is related to the execution time of the reference superscalar DSP architecture. From the performance results, we can see that, as expected, the optimized version of code performs twice faster than the vectorizable code. On the other hand, we can also see that the long vector architecture is not cost effective, as it is hardly able to outperform the short vector architecture (except for `djpeg` which exhibits a 10% of performance improvement).

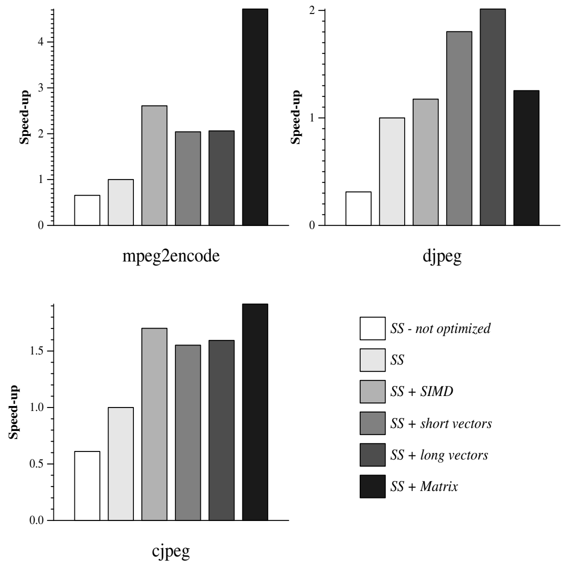

Figure 7. Performance results.

In `mpeg2encode`, the SIMD architecture outperforms both vector architectures leveraging a 25% of additional performance while the matrix architecture provides a huge performance boost of 2.2X. Two reasons explain this performance gains: (a) the smaller number of overall operations to execute, and (b) the higher parallelism (for instance, the SIMD version executes 6.6 vector operations per cycle while the vector architectures execute only up to 4 vector operations per cycle).

Neither of the two previous reasons can explain why the SIMD and the matrix architectures slightly outperform the conventional vector architectures (15% and 20% respectively) in `cjpeg`. As we will see in section 4.5, the pack/unpack overhead in those architectures (basically, to perform matrix transpose and *RGB* color restructuring) allow to make full 64-bit vector memory accesses. On the contrary, the conventional vector architecture find the vector elements not consecutively arranged in memory thus leveraging very poor data bandwidth.

On the other hand, the problem of the data rearrangement is not encountered in `djpeg`, since the *IDCT* is not vectorized at all, and the inverse *color conversion* algorithm has the color dimensions adequately rearranged. Moreover, the matrix architecture cannot take benefit from the multimedia accumulators in the *upsample* algorithm (thus needing costly data promotion transformations). As a result, due to the difference in the number of operations to execute, the conventional vector architectures provides a performance benefit over the SIMD and matrix architectures of a 50% for the short vector architecture and a 66% for the long vector architecture.

Speed-up	SS	SIMD	SV	LV	Matrix
mpeg2encode	24.2 %	31.1 %	63.3 %	59.5 %	32.6 %
djpeg	29.1 %	30.5 %	43.6 %	37.4 %	27.2 %
cjpeg	50.4 %	37.2 %	51.0 %	45.6 %	33.3 %

Table 2. Impact of out-of-order execution.

4.4 The impact of out-of-order execution

Historically, out-of-order execution has been considered not an option for the embedded domain. Theoretically, if the performance gains were not higher than the obvious increase in power consumption, an out-of-order execution engine would not be worth. Table 2 shows some speed-ups when using a very limited form of out-of-order rather than the baseline architecture. Note that the Speed-ups are relative to the correspondent in-order architecture. From the results, we can see that out-of-order execution provides a considerable performance boost for all the architectures (between 30% and 60%). Nevertheless, it still remains to be seen whether these performance gains are higher than the increase in power consumption.

Contrary to our expectations, the architectures that benefit the most from an aggressive dynamic scheduling are the conventional vector architectures. From the architectural point of view, the streaming nature of the vector instruction should leverage a convincing throughput even for in-order execution. However, looking carefully at the behavior of the pipeline for in-order and out-of-order execution we can easily figure out what produces such a performance degradation in the basic in-order approach. Our vector architecture model does not allow memory chaining (that is, a vector arithmetic operation does need to wait until a dependent vector load has executed totally). This design decision was due to the fact that, while memory chaining is worth for systems with fixed memory latency, having a cache hierarchy (and thus, unpredictable latencies) complicates too much the control logic of the in-order processor. As a consequence, when we encounter a load-arithmetic dependence, the processor stalls many cycles waiting for an entire long stream of data.

The lack of memory chaining is not much of a problem if we consider a careful static scheduling. By exploiting at compile time the available ILP so that we can separate the further possible a vector load from a dependent vector arithmetic instruction, we can perform statically what out-of-order execution does. Unfortunately, the CONVEX compiler does not consider these issues and tends to put together dependent load-arithmetic instructions. In sharp contrast with this poorly scheduled code, the MMX-like SIMD and the matrix architectures were carefully tuned exploiting most of the potential ILP at the static scheduling level, so taking little benefit from a more aggressive dynamic execution.

Slow-down	SS	SIMD	SV	LV	Matrix
mpeg2encode	-5 %	-16 %	-14 %	-18 %	-18 %
djpeg	-15 %	-13 %	-13 %	-10 %	-12 %
cjpeg	-23 %	-6 %	-19 %	-21 %	-12 %

Table 3. Memory performance degradation.

Hit rate	SS	SIMD	SV	LV	Matrix
mpeg2encode	98.3 %	95.5 %	97.3 %	94.4 %	94.6 %
djpeg	96.6 %	95.2 %	92.8 %	89.4 %	92.4 %
cjpeg	93.7 %	97.6 %	93.3 %	93.0 %	93.5 %

Table 4. Data cache hit rate.

4.5 Memory behavior

Table 3 shows the performance degradation when we consider a realistic memory system (relative to the performance with a perfect cache). The effect of the instruction cache over performance has demonstrated to be very small, as all programs fit perfectly in cache. Therefore, we are going to focus on the impact of the data cache.

From the results on table 3 we can see three interesting facts. First, even with high hit-ratios (see table 4), considerable performance losses around 15% are produced when considering a realistic memory system. This is due to the fact that we do not have a L2 cache, and for all the accesses that miss in the first level we must pay all the full latency of the external memory sub-system. Another interesting fact is that the MMX-like SIMD architecture seems to be the most robust alternative to the impact of the memory system.

Additionally, we can observe that the vector architectures (short, long and matrix) exhibit the highest performance degradations when considering a realistic memory system. This fact may seem counterintuitive, since vector architectures are very well known for their capability to tolerate memory latency along long streams of data. However, this advantage is fully compensated by the fact that as we have longer vectors the data locality degrades (as it can be seen in table 4, where we show the data cache hit rates). The reason resides in the way that the different architectures execute a loop. A non-vector architecture executes all the instructions of a loop iteration in a sequential way, exploiting temporal locality. If there is spatial locality, a miss access produces an effect of 'prefetching' that loads the elements of the following iterations. This effect produces a natural way of exploiting spatial locality. On the other hand, vector architectures execute several loop instances of the same instruction sequentially, thus not being able to exploit this advantage and finding less elements in the cache.

As a matter of fact, the higher the vector length the lower the hit rate. On the other hand, the higher the vector length the higher the tolerance to memory latency. Therefore, this two effects compensate together. For instance, the long

Latency	SS	SIMD	SV	LV	Matrix
mpeg2encode	1.61	2.64	2.46	2.85	3.26
djpeg	2.14	2.33	2.07	1.83	2.05
cjpeg	3.36	1.48	2.78	2.86	1.91

Table 5. Average memory latency.

vector architecture leverages very high vector lengths for `djpeg`, and despite the fact that the hit-ratio is very low (a 89.4%), the performance loss (only a 10%) and the average memory latency are the lowest for all the architectures. This effect is not encountered in the basic superscalar architecture and, as a result, low hit-ratios or even moderate latencies often translate into a high performance degradation, as it can be seen in table 5 for the `cjpeg` benchmark.

The average memory latency is not only dependent on the data locality. For instance, for `cjpeg`, the conventional vector architectures have higher latencies than the matrix architecture having, though, the same data locality. This is explained by the fact that in `cjpeg` we find a high percentage of vector accesses whose elements are not consecutively arranged in memory (for example, when accessing the color dimension we find stride 3, or when traversing columns in the IDCT we find stride 8). As a result, the memory system is forced to access the elements individually, leveraging very poor effective bandwidth (which ends up affecting the overall latency of the vector access). This explains why the conventional vector architectures do not outperform the SIMD architecture, even though they execute fewer operations. The SIMD and matrix architectures do not face this problem as they use logic operations to rearrange the data conveniently.

5 Architectural complexity comparison

Table 6 compares the register file configuration and the overall size for all the architectures under study. Note that due to the simplicity of the basic processor core (same fetch/decode/issue rate, fixed number of register file and cache ports), the only parameters that influence the final area cost are the number and wide of the functional units and the size of the register files. Since the number of register file ports is fixed, the area cost is proportional to the register file size.

Therefore, from the point of view of performance/cost ratio, we can see that the in-order SIMD architecture is the most cost effective approach. The SIMD architecture outperforms the conventional vector architectures for two of the benchmarks, despite having one single 64-bit functional unit (instead of 4 32-bit functional units) and a overall register file size between 3 and 22 times smaller. On the other hand, the matrix architecture could be a good alternative when

	int	vector	accum.	Total
SS	32 x 32b	0 x 1 x 64b	0 x 192b	0.125 KB
SS (ooo)	40 x 32b	0 x 1 x 64b	0 x 192b	0.156 KB
SS + SIMD	32 x 32b	32 x 1 x 64b	0 x 192b	0.375 KB
SS + SIMD (ooo)	40 x 32b	40 x 1 x 64b	0 x 192b	0.469 KB
SS + SV	52 x 32b	16 x 16 x 32b	0 x 192b	1.203 KB
SS + SV (ooo)	64 x 32b	24 x 16 x 32b	0 x 192b	1.750 KB
SS + LV	52 x 32b	16 x 128 x 32b	0 x 192b	8.203 KB
SS + LV (ooo)	64 x 32b	24 x 128 x 32b	0 x 192b	12.25 KB
SS + matrix	32 x 32b	16 x 16 x 64b	2 x 192b	2.172 KB
SS + matrix (ooo)	40 x 32b	20 x 16 x 64b	4 x 192b	2.750 KB

Table 6. Overall register file sizes.

looking for high performance, since it provides the best performance at a reasonable area cost (about 6 times larger that the MMX-like SIMD architecture register file size).

On the other hand, the long vector architecture does not arise as a good option, since requires a register file size of the same order of the data and instruction caches while is hardly able to provide performance gains when comparing with the short vector architecture.

6 Summary

The focus of this paper has been to provide some quantitative data in order to understand the performance/cost trade-offs of different DSP exploitation alternatives for the embedded domain. We have selected three entire benchmarks from the *Mediabench* suite and we have evaluated them with different SIMD/vector architectures.

We have also analyzed the differences between the explicit parallel versions of code, based on the standard algorithms, and the high-tuned non-vectorizable original versions of code, demonstrating that the optimized version can execute up to two times faster that the non-optimized one.

From the performed quantitative analysis, we conclude that the SIMD-like architecture arises as the more cost-effective option, as it can provide convincing performance (with gains over the reference DSP ranging from 1.2x to 2.6x) with only 32 64-bit registers in front of the nearly six times larger register file needed by the matrix architecture. Nevertheless, if we are looking for maximum performance, the latter can be considered, since it outperforms all the other architectures at a reasonable cost. As far as vector architectures are concerned, it has been demonstrated that a long vector processor is not worth at all for the embedded domain, as it does not achieve enough performance gains (only a 10% over short vectors for one of the benchmarks) to justify the enormous cost increase. However, short vectors could be a good option for some media codes (with few reduction operations and conveniently arranged data) if the cost of out-of-order execution is acceptable.

We have also seen that two effects compensate together in vector architectures. Vector streams exhibit worse data locality which translates into lower hit rates. Nevertheless, the performance impact of this effect is compensated by the latency tolerance capability typical of vector memory accesses. On the other hand, scalar architectures do not tolerate well increases in the latency of the memory system.

SIMD-like architectures need the data to be consecutively arranged in memory. Therefore, they have a huge overhead involving data restructuring (such as matrix transpose). We have observed that this overhead may end up being beneficial in some cases (since allow to maximize the data cache bandwidth) while in other cases is detrimental to performance.

This paper has focused on superscalar architectures exploiting different levels of DLP. Nevertheless, a very promising alternative is the inclusion of SIMD-like instructions in VLIW embedded processors. This alternative appears as a good match for the matrix architecture, since it is able to exploit one of the dimensions of parallelism by means of DLP (SIMD instructions) while it is able to exploit the other by means of ILP (by using wide-issue static scheduling).

References

[1] Keith Diefendorff. Sony's emotionally charged chip. *Microprocessor report*, pages 6–11, April 1999.

[2] Manfred Schlett. Trends in embedded microprocessor design. *IEEE Computer*, pages 44–49, August 1998.

[3] Alex Peleg and Uri Weiser. MMX Technology Extension to the Intel Architecture. *IEEE Micro*, pages 42–50, August 1996.

[4] Marc Tremblay, J. Michael O'Connor, Venkatesh Narayanan, and Liang He. VIS Speeds New Media Processing. *IEEE Micro*, pages 10–20, August 1996.

[5] Mips extension for digital media with 3d. Technical Report http://www.mips.com, MIPS technologies, Inc., 1997.

[6] Semiconductor Group Siemens. Tricore. 32-bit single-chip uc-dsp. *Technical Summary*, http://www.tri-core.com/, 1998.

[7] Philips Semiconductors. Trimedia tm-1300. http://www-us3.semiconductors.com/trimedia/, 1999.

[8] Analog Devices. Introducing tigersharc. *Whitepaper*, http://www.analog.com/new/ads/html/SHARC2.

[9] TI. TMS320C62XX family. Technical Report http://www.ti.com/sc/docs/products/dsp/tms320c6201.html, Texas Instruments, 1999.

[10] Corinna G. Lee and Derek J. DeVries. Initial results on the performance and cost of vector microprocessors. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 171–182, Research Triangle Park, North Carolina, December 1–3, 1997. IEEE Computer Society TC-MICRO and ACM SIGMICRO.

[11] Corinna G. Lee and Mark G. Stoodley. Simple Vector Microprocessors for Multimedia Applications. In *Proceedings of the 31st Annual International Symposium on Microarchitecture*, pages 330–335, Dallas, Texas, December 1998. IEEE Computer Society TC-MICRO and ACM SIGMICRO.

[12] Krste Asanovic et. al. The to vector microprocessor. *Hot Chips*, VII:187–196, August 1995.

[13] Christoforos Kozyrakis and David Patterson. A new direction for computer architecture research. *IEEE Computer*, pages 24–32, November 1998.

[14] William J. Dally. Tomorrow's computing engines (Keynote Speech). In *HPCA-4*, February 1998.

[15] Francisca Quintana, Jesus Corbal, Roger Espasa, and Mateo Valero. Adding a vector unit on a superscalar processor. *International Conference on Supercomputing*, Available at http://www.ac.upc.es/homes/roger/papers/list.html, June 1999.

[16] Jesus Corbal, Roger Espasa, and Mateo Valero. Exploiting a new level of dlp in multimedia applications. *MICRO*, 1999.

[17] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 330–335, Research Triangle Park, North Carolina, December 1–3, 1997. IEEE Computer Society TC-MICRO and ACM SIGMICRO.

[18] Gregory K. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, April 1991.

[19] Thomas Sikora. *MPEG Digital Video Coding Standards*. McGraw W-Hill Book Company, Berlin, 1995.

[20] Byeong Gi Lee. A new algorithm to compute the discrete cosine transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-32, 6:1243–1245, Dec. 1997.

[21] Amitabh Srivastava and Alan Eustace. ATOM: A system for building customized program analysis tools. *SIGPLAN Notices*, 29(6):196–205, June 1994. *Proceedings of the ACM SIGPLAN '94 Conference on Programming Language Design and Implementation*.

[22] Pentium iii processor: Developer's manual. Technical Report http://developer.intel.com/design/PentiumIII, INTEL, 1999.

[23] Jesus Corbal, Roger Espasa, and Mateo Valero. Mom: Instruction set architecture. Technical report, Universitat Politècnica de Catalunya, 1999.

[24] Parthasarathy Ranganathan, Sarita Adve, and Norman P. Jouppi. Performance of image and video processing with general-purpose and media isa extensions. *International Symposium on Computer Architecture*, May 1999.