Scalability and Complexity of 2-Dimensional SIMD Extensions

Mauricio Alvarez, Friman Sánchez, Esther Salamí, Alex Ramirez and Mateo Valero

Abstract— **Current SIMD extensions have probed to be effective for incrementing the performance of general purpose and embedded microprocessors for multimedia applications, but they seem to be not capable of scale their performance as required by new multimedia standards and applications. In this paper we describe and evaluate the scaling of a SIMD extension based on matrix registers and present a performance comparison with the scaling of SIMD extensions used in current microprocessors. This matrix architecture adapts well to the data structures used in image, video and audio applications and can be incorporated without bigger modifications into a high performance superscalar processor. Evaluations have demonstrated a significant performance improvement over conventional SIMD extensions. Speed-up over a 2-way superscalar processors with MMX-like extension goes up to 3.3X for complete applications. The reduced instruction count, that reduces the pressure in the front end, and the distributed nature of the register file make the matrix extension a complexity-effective solution suitable for the next generation of high performance processors.**

Key words— **Computer architecture, SIMD, multimedia extensions, vector processors.**

I. Introduction

MULTIMEDIA applications have become one of the most important workloads in contemporary computing [1]. In general purpose processor the common approach for dealing with the requirements of multimedia applications has been the extension of the Instruction Set Architecture with SIMD instructions that exploit data level parallelism (DLP) by performing multiple sub-word operations in a single register [2].

As multimedia standards become more complex, processors need to scale their SIMD multimedia extensions in order to support new computing demanding features. One way of scaling is making registers wider [3] in order to pack more data into a single register. In this way is possible to process more operations within a single instruction but it is not clear that future microprocessors may scale the SIMD extensions from current 128-bit registers to 256-bit or more, mainly because the significant overhead that would be needed for packing and unpacking data [4]. Another way of scaling is to add more SIMD functional units to the corresponding pipeline, but some studies [5] have demonstrated that SIMD execution units are underutilized because of bottlenecks that appear in the non-SIMD portion of the pipeline.

On the other hand, some research proposals suggest the use of vector processors [6], [7] as an effective way of exploiting DLP present in multimedia applications. An alternative approach comes from the combination of vector registers and sub-word computation in such a way that registers can be seen as matrices [8].

In this paper a Matrix Oriented Multimedia Architecture (MOM) with augmented registers is presented as an effective way of processor scaling for multimedia applications. We show that making these matrix registers bigger, it is possible to pack almost all the data available in the inner loops of common multimedia kernels. By the use of matrix registers, the dynamic instruction count is reduced drastically while still allowing the execution of more operations in parallel. The final results show a significant increment of performance in the evaluated multimedia applications. Further extensions in width or length of SIMD registers would not result in significant performance improvements.

This paper is organized as follows: In chapter 2 the differences between scaling one-dimensional and two-dimensional extensions are presented and a comparison of complexity issues of both schemes is also detailed. In chapter 3 the applications, simulator and modeled microarchitectures used in the experiments are described. In chapter 4 performance results for both kernels and complete applications are analyzed. Finally in chapter 5 some conclusions are presented.

II. SIMD and Matrix Extensions for Multimedia

A. Scaling 1-Dimensional SIMD Extensions

As it was mentioned, there are two main ways for scaling current multimedia extensions. The first approach is scaling the resources of a superscalar processor. Such an approach has two main disadvantages: First, there is the growing complexity of critical structures in the pipeline that can make unfeasible a high aggressive superscalar processor with many SIMD functional units; and, second, even if such a processor could be developed, performance gains could not be as good as expected. Recent studies [5] suggest that there are some bottlenecks in the microarchitecture that do not allow to obtain better performance by resources scaling. These bottlenecks are related with overhead and supporting instructions necessary for address arithmetic, data transformation, access overhead and branches.

Increasing the width of SIMD registers, from cur-

Computer Architecture Department. Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. {fsanchez, alvarez, esalami, aramirez, mateo}@ac.upc.es. This work has been supported by the Ministry of Science and Technology of Spain, the European Union (FEDER funds) under contract TIC2001-0995-C02-01 and by an IBM Faculty Award to prof. Mateo Valero. We acknowledge the European Center for Parallelism of Barcelona (CEPBA) for supplying the computing resources for our research.

rent 128-bit registers to 256-bit, 512-bit or more, is the other way of scaling. The idea is to pack more data into a single register and then to perform more operations in parallel. But the majority of image, video an audio applications use data in small arrays or matrices sometimes non-contiguous in memory. For this kind of applications, making registers bigger than the basic data structures may incur a big overhead for taking data from memory and/or storing back the results. Additionally the amount of data that can be packed is always fixed by the register width and changes to it implies a big source code modification.

B. Scaling the Matrix Extension

MOM is a matrix-oriented ISA paradigm for multimedia applications, based on fusing conventional vector ISAs with SIMD ISAs such as MMX. MOM is a suitable alternative for the multimedia domain due to its efficiently handling of the small matrix structures typically found in most multimedia kernels [8].

The register file organization proposed in the original MOM architecture [9] provides the programmer with 16 matrix registers, each one holding 16 64-bit words. In this work we study how vector register file in MOM architecture can scale from 64-bit (referred as VMMX64) to 128-bit (referred as VMMX128) registers. Additionally we compare the matrix extension scaling with the benefits and limitations of scaling a MMX-like extension with 64-bit registers (MMX64) to a 128-bit extension like Intel SSE2 (MMX128).

Figure 1 shows four different areas that represent the storage capacity of the studied approaches. Area (1) represents the original 64-bit register in MMX alternative and area (2) shows how MMX64 scales to MMX128.

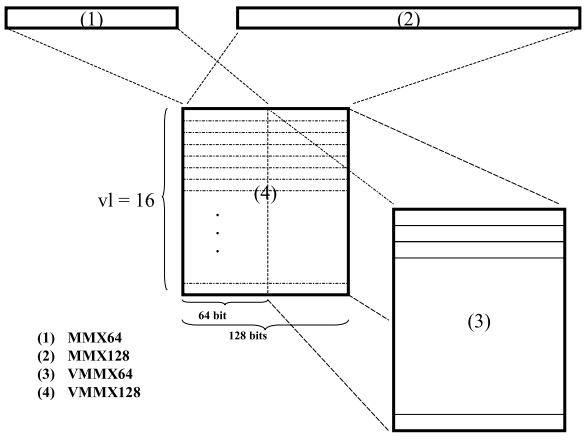

Fig. 1. Register capacity

Area (3) shows the capacity of a vector register in VMMX64. VMMX64 overcomes the limitation of MMX64, and even MMX128, allowing a maximum vector length of 16, which it is enough for common multimedia algorithms. Finally, Area 4 shows, the way in which VMMX64 scales to VMMX128 allowing to pack up to 16x16 bytes in a single register. This is particularly useful for image and video applications.

When scaling the VMMX version from 64-bit to 128-bit registers a modification in the ISA was necessary in order to improve the support for the 64-bit data type. Instructions like partial load or store were added. These instructions are important for applications that can not take advantage of the full 128-bit registers due to their limited data parallelism.

C. Complexity issues when scaling multimedia register files

In order to reduce the complexity of the register file in VMMX configurations, functional units can be replicated and each matrix register can be splitted across lanes [10]. Figure 2 shows an implementation of this approach. As an example, in the 4-way MMX configuration, a centralized register file connected to 4 arithmetic units needs 12 read ports and 8 write ports. This centralized register file that interconnects every arithmetic unit is costly because it provides both storage for and communication between arithmetic units in such a way that any ALU can read from or write to any storage location.

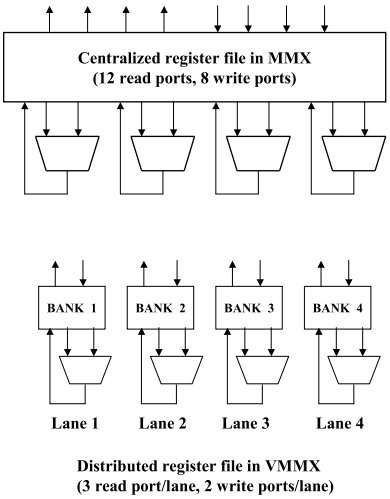

Fig. 2. Register File Comparison.

On the other hand, 4-way VMMX configuration has 2 functional units, each one with 4 lanes, as figure 2 shows. For room reasons, we only show one of the functional units. The register file is divided into 4 banks across lanes, each bank has 3 read ports and 2 write ports. Although this organization is limited in connectivity, it is less complex than centralized organization. In this case, the restricted communication can be made in a way that matches the parallelism and access patterns of multimedia applications. The distributed register file organization is a more cost-effective alternative than the centralized approach [11], in terms of area, time, and power.

III. Experimental Methodology

A. Workload

In order to evaluate the different architectures under study we have selected six applications from the Mediabench suite [12] that are representative of

video, still image and voice processing applications. For each application we have selected the most computational intensive kernels with potential DLP and evaluated them in isolation. Tables I and II describe the kernels and benchmarks and their characteristics.

TABLA I
KERNEL SET DESCRIPTION

Kernel	Description, (used in)
idct	Discrete Cosine Transform (*mpeg2enc,jpegenc*)
motion1	Motion estimation (SAD) (*mpeg2enc*)
motion2	Motion estimation (SQD)) (*mpeg2enc*)
comp	Motion compensation (*mpeg2dec*)
addblock	Picture decoding (*mpeg2dec*)
rgb	RGB to YCC color conversion (*jpegdec*)
ycc	YCC to RGB color conversion (*jpegenc*)
h2v2	Image up-sampling (*jpegdec)*
ltppar	Parameter calculation for LTP filtering (*gsmenc*)
ltpfilt	Long term parameter filtering (*gsmdec*)

TABLA II
BENCHMARK SET DESCRIPTION

Application	Description	Input data set
jpegenc	JPEG still image encoder	penguin.ppm
jpegdec	JPEG still image decoder	penguin.jpg
mpeg2enc	MPEG2 video encoder	mei16v2rec
mpeg2dec	MPEG2 video decoder	mei16v2rec.mpg
gsmenc	GSM 06.10 speech encoder	clinton.pcm
gsmdec	GSM 06.10 speech decoder	clinton.gsm

B. Simulation Framework

Emulation libraries containing the multimedia instructions have been used for the four extensions evaluated: MMX64, MMX128, VMMX64 and VMMX128. Most of the functionality of MMX and SSE ISAs have been implemented into the MMX64 and MMX128 emulation libraries respectively, but is important to note that the MMX libraries do no exactly model the MMX or SSE instruction set. Using these emulation libraries, optimized versions of the mentioned kernels were developed. For maximizing performance, common optimization techniques like loop-unrolling and software pipelining were applied. In all cases, the baseline ISA is the Alpha ISA. In order to generate code for it, gcc version 2.95.2 was used. All codes have been compiled using the -O flag.

The simulation tool used in this work was an improved version of Jinks Simulator [13], that is a parametrizable simulator targeted at evaluating superscalar architectures with a special focus on vector extensions. An execution-driven approach based on ATOM [14] was used for generating the input trace stream for the simulator.

C. Processor Models

The baseline processor is a 2-way out-of-order superscalar core similar to MIPS R10000 [15] with the addition of a MMX64 SIMD extension. We have evaluated four different configurations that include MMX and VMMX approaches for 64 and 128-bit registers.

- 2/4/8-way superscalar processor + MMX64
- 2/4/8-way superscalar processor + MMX128
- 2/4/8-way superscalar processor + VMMX64
- 2/4/8-way superscalar processor + VMMX128

Table III shows the processor configurations used for simulations. Although the 8-way superscalar processor with SIMD units is a very aggressive configuration that is nowadays unfeasible in a high performance general purpose processor at current clock frequencies, it can be used as a guide of the potential performance that could be obtained scaling processor resources.

TABLA III
MODELED PROCESSORS

Parameter	MMX 2/4/8 way	VMMX 2/4/8 way
Fetch, Decode, Grad.	2/4/8	2/4/8
Integer FUs	2/4/8	2/4/8
FP FUs	1/2/4	1/2/4
SIMD issue	2/4/8	1/2/3
SIMD FUs	2/4/8	1/2/3
Mem FUs (64b ports)	1/2/4	1/1/2
L2 ports	64b/128b/256b	64b/128b/256b
Logical Media registers	32	16x16
Physical Media registers (MMX or VMMX)	40/64/96	20/36/64
Lanes	1	4
Banks per Lane	1	1/2/4
Read ports per Bank	6/12/24	3
Write ports per Bank	4/8/18	2

A detailed memory hierarchy model with two levels of on-chip cache and a Direct RAMBUS main memory system have been included in the simulator. Table IV shows the configuration parameters for caches and main memory. For VMMX versions a *vector cache* was used [16]. The *vector cache* is a two-bank interleaved cache targeted at accessing stride-one vector requests by loading two whole cache lines (one per bank) instead of individually loading the vector elements. Then, an interchange switch, a shifter, and a mask logic correctly align the data. Scalar accesses are made to the L1 conventional data cache, while vector accesses bypass the L1 to access directly the L2 vector cache. If the L2 port is $B\times$ 64-bit wide, these accesses are performed at a maximum rate of B elements per cycle when the stride is one, and at 1 element per cycle for any other stride. A coherency protocol based on an exclusive-bit policy plus inclusion is used to guarantee coherency.

TABLA IV
MEMORY HIERARCHY CONFIGURATION (WT=WRITE-THOUGH, WB=WRITE-BACK, NWA=NO-WRITE-ALLOCATE, WB DEPTH/RETIRE = WRITE BUFFER ENTRIES/RETIRE

	L1	L2
size	32KB	512KB
number of ports	1/2/4	1
port width (bytes)	8	16/32/64
number of banks	8	2
sets per bank	32	2048
associativity	4	2
line size (bytes)	32	128
write policy	WT	WB
allocate policy	NWA	NWA
latency	3	12
MSHR entries	8	8
WB depth/retire	8/4	8/4
Main Memory Latency (cycles)	500	

IV. Simulation results

A. Kernel speed-up analysis

Figure 3 shows the kernels speed-up for the different multimedia ISAs under study. The baseline is a 2-way superscalar processor with a MMX64 extension.

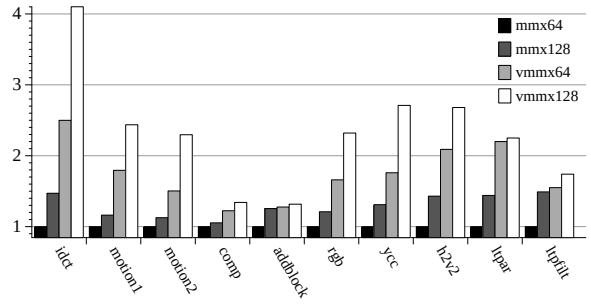

Fig. 3. Kernels speed-up (2-way)

Scaling from MMX64 to MMX128 does not result in great performance improvements taken into account that registers and functional units are twice the size of the MMX64 ones. The speed-up only goes up to 1.47X for *idct*, 1.43X for *ycc*, 1.25X for *addblock* and 1.19X for *h2v2*. These kernels have the most regular data patterns and they adapt well to 128-bit wide registers. The average speed-up for MMX128 kernels is 1.24X.

Results also show that both VMMX versions exhibit bigger speed-ups than the MMX ones in all the cases and produce significant speed-ups when going from VMMX64 to VMMX128 versions, except for *ltppar* and *addblock* kernels. The bigger speed-ups of kernels (4.10X for *idct*, 2.71X for *ycc*, 2.68X for *h2v2*, 2.43X for *motion2* and 2.29X for *motion1*) are due to the matching between the data organization of these kernels with the matrix registers structure. This include: higher vector length (*motion1* and *motion2*) that allows a full utilization of vector registers, a stride of one that allows to find consecutive data in memory (*h2v2*) and the use of vector registers as a cache for intermediate results (*idct*).

The small speed-up obtained by *comp* and *addblock* in both MMX128 and VMMX128 versions is related with the small parallel data available (8x4 pixels in *comp* with a stride of 800) that only represents a small fraction of the matrix registers.

B. Complete applications speed-up

Speed-up of kernels only shows the potential of the evaluated architectures on a highly data parallel code, but these kernels are used in bigger applications where there is a lot of scalar code that can not use the SIMD units. Then the multimedia extensions proposed not only need to exploit efficiently the data parallelism available but not to degrade the performance of the scalar portion of the application.

Mpeg2enc is the application which takes more benefit from the use of matrix registers. Figure 4 shows that VMMX versions of the application scales much better than MMX ones. VMMX128 version has the biggest speed-up due to the good matching of data in the *motion* and *idct* kernels to the 128-bit matrix registers. These kernels account for more than 80 percent of the total execution time of the whole application. *Mpeg2dec* instead shows a significant speed-up but the difference between MMX and VMMX versions are smaller than in *mpeg2enc*. In this application motion compensation routines are not so much significant of the total execution time and their data parallelism is not so big. Furthermore *mpeg2dec* presents a lot of scalar code in picture decoding that can not be vectorized.

In *jpegdec* application, VMMX versions show a greater capacity to exploit DLP compared with MMX versions. This is due to the fact that *h2v2* and *ycc* kernels operate over data that has a good pattern in memory and the vector length used is high (8 and 16 in both cases). On the other hand, VMMX64 version of *jpegenc* obtains a better performance than MMX64 and MMX128 for less aggressive schemes, that is 2 and 4-way. However, in 8-way configuration, MMX128 version outperforms the VMMX64 version, this is due to the behavior of *rgb* kernel. The vectorization happens along the color space (Red, Green and blue) dimension, yield a vector length of only 3. Additionally, the order in which results must be stored in memory does not benefit the VMMX64 version. However, the VMMX128 version overcomes this limitation allowing to pack of more sub-word data into the matrix register.

In *gsmenc* and *gsmdec* applications, DLP is small due to the dominance of the scalar code as explained in the kernels section.

As showed in figure 4, for *mpeg2enc*, a similar performance can be obtained by using a 2-way VMMX128 processor instead of a 8-way MMX128 one. The same behavior can be seen in *jpegenc* and *mpeg2dec* between 4-way VMMX versions and 8-way MMX ones. In those cases, scaling the register file of the 2-dimensional architecture is much more effective than scaling the complete resources of a processor with 1-dimensional registers. As can be derived the complexity of a 2way superscalar processor with a 2-dimensional extension is smaller that a 8-way processors with a MMX-like extension.

C. Cycle Breakdown

Figure 5 shows the dynamic cycle distribution for the *jpegdec* application. The remaining of the benchmarks are not included for room reasons, but they exhibit a similar behavior. The shadow part of each column represents the dynamic cycles used in vector operations, while the white part comes from the scalar code. Results are normalized by the dynamic cycle count of the reference 2-way MMX64 superscalar processor.

As it was expected, scaling the MMX64 extension provides a significant drop in the number of cycles to execute the vector part of code. Scaling in both dimensions (width and length) achieves the maximum reduction: for the 2-way architecture, the VMMX128

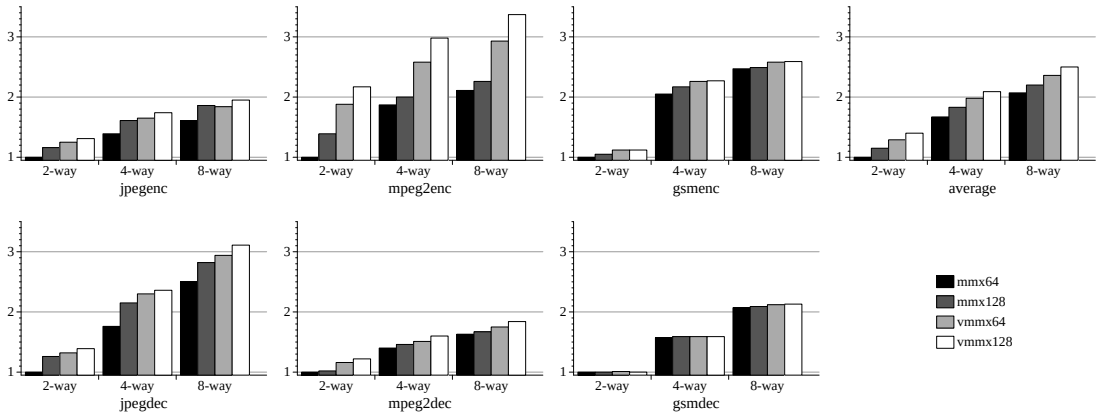

Fig. 4. Full applications speed-up

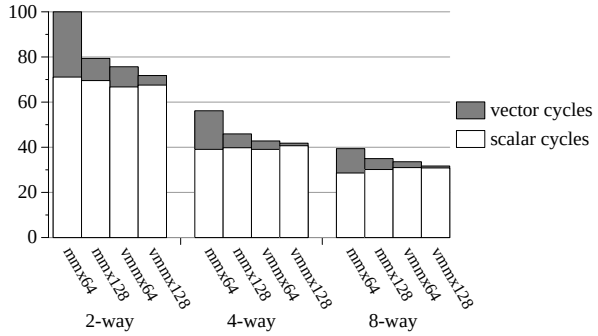

Fig. 5. Cycle count distribution (jpegdec)

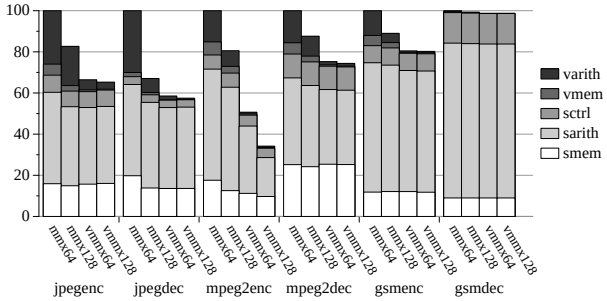

Fig. 6. Dynamic instruction count

extension reduces the execution time of the vector code in a 85% over the MMX64 extension.

However, it can be observed that, when most of the available DLP parallelism is exploited via multimedia extensions, the remaining scalar part of the code becomes the main bottleneck. For the 8-way VMMX128 architecture, the vector cycles only represent the 2.7% of the overall execution time. By the Amhdal Law, further improvements in the execution of the vector region would be imperceptible in the full application.

D. Dynamic Instruction Count

Figure 6 shows the dynamic instruction count for the benchmarks under study. Again, results are normalized by the dynamic instruction count of the MMX64 architecture. The operations have been classified into five categories: scalar memory, scalar arithmetic, control, vector memory and vector arithmetic. We observe that the VMMX architectures execute about 30% fewer instructions than the MMX64, and the MMX128 an average of 15% fewer instructions. This is obviously due to the capability of these extensions to pack more operations into a single instruction.

As seen in figure 6, the biggest instruction reduction is achieved by the *mpeg2enc* application. This reductions comes from the commented elimination of scalar instructions used for address computation and loop manipulation. In any way, note that the limit of packing data seems to be reached, and scaling further over, either in width or in length, would not provide any noticeable benefit. At this point, a full reorganization of the code is required if we want to expose more parallelism to the processor.

V. Conclusions

In this paper we have presented an scalability analysis of SIMD extensions for multimedia applications. Scaling current 1-dimensional SIMD extensions was compared to scaling a 2-dimensional architecture. The comparison was made using both kernels and complete multimedia applications. Scaling was made in the width of SIMD registers and in processor resources. The matrix architecture with 128-bit registers has shown the best performance improvements compared to a 64-bit matrix architecture and one-dimensional(64-bit and 128-bit) SIMD extensions. The benefits of a Matrix architecture come from the elimination of some of the bottlenecks of current SIMD extensions. Multimedia data structures fit very well into matrix registers, the matrix nature of the ISA eliminates a lot of pointer and loop code overhead and the combination of vector length and vector stride eliminate the constraints in the contiguous data distribution in memory. In some cases additional performance improvements can be obtained by the use of matrix registers as a cache for intermediate results reducing also the pressure on the memory hierarchy.

By applying all of these optimizations, the Matrix architecture is reaching the limits of available DLP in the inner loops of common multimedia applications. Further scaling on the width or length of matrix registers can no deliver significant performance improve-

ments because the execution time is now dominated by the scalar portion of the code. Extracting more parallelism in the analyzed applications requires special code transformations in order to execute multiple instances of the optimized functions in parallel or dedicated hardware to extract parallelism beyond the inner loops.

References

[1] K. Diefendorff and P.K. Dubey, "How multimedia workloads will change processor design," *IEEE Micro*, vol. 30, no. 9, pp. 43–45, September 1997.

[2] N. T. Slingerland and A. J. Smith, "Multimedia instruction sets for general pupose microprocessors: A survey," Tech. Rep. CSD-00-1124, UCB, December 1999.

[3] 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, August 2000.

[4] S. Larsen and S. Amarasinghe, "Exploiting superword level parallelism with multimedia instruction sets," in *Proceedings of the SIGPLAN '00 Conference on Programming Language Design and Implementation*, June 2000.

[5] Deepu Talla, Lizy Kurian John, and Doug Burger, "Bottlenecks in multimedia processing with simd style extensions and architectural enhancements," *IEEE Transactions on Computers*, vol. 52, no. 8, pp. 1015–1031, August 2003.

[6] K. Asanovic, J. Beck, B. Irissou, B. Kingsbury, N. Morgan, and J. Wawrzynek, "The t0 vector microprocessor," in *Hot Chips VII*, August 1995, pp. 187–196.

[7] C.E Kozyrakis and D.A. Patterson, "Scalable vector processors for embedded systems," *IEEE Micro*, vol. 23, no. 6, pp. 36–45, Nov–Dec 2003.

[8] Jesus Corbal, Roger Espasa, and Mateo Valero, "Exploiting a new level of dlp in multimedia applications," in *32nd international symposium on Microarchitecture*, 1999, pp. 72–79.

[9] Jess Corbal, *N-Dimensional Vector Instruction Set Architectures for Multimedia Applications*, PhD thesis, UPC, Departament d'Arquitectura de Computadors, 2002.

[10] Krste Asanovic, *Vector Microprocessors*, PhD thesis, University of California at Berkeley, 1998.

[11] Scott Rixner, William J. Dally, Brucek Khailany, Peter Mattson, Ujval J. Kapasi, , and John D. Owens, "Register organization for media processing," in *Tenth International Symposium on High Performance Computer Architecture*, January 2000.

[12] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems," in *30th International Symposium on Microarchitecture*, 1997, pp. 330–335.

[13] Roger Espasa, "Jinks: A parametrizable simulator for vector architectures," Technical Report UPC-CEPBA-1995-31, Universitat Politécnica de Catalunya, 1995.

[14] Amitabh Srivastava and Alan Eustace, "ATOM: A system for building customized program analysis tools," *ACM SIGPLAN '94 Conference on Programming Language Design and Implementation*, vol. 29, no. 6, pp. 196–205, June 1994.

[15] K. C. Yeager, "The mips r10000 superscalar microprocessor," *IEEE Micro*, vol. 16, no. 2, April 1996.

[16] Francisca Quintana, Jesus Corbal, Roger Espasa, and Mateo Valero, "Adding a vector unit on a superscalar processor," in *International Conference on Supercomputing*, June 1999, pp. 1–10.