Three-Dimensional Memory Vectorization for High Bandwidth Media Memory Systems

Jesus Corbal, Roger Espasa and Mateo Valero

Departament díArquitectura de Computadors,

Universitat Politècnica de CatalunyañBarcelona, Spain *e-mail: {jcorbal,roger,mateo}@ac.upc.es

Abstract

Vector processors have good performance, cost and adaptability when targeting multimedia applications. However, for a significant number of media programs, conventional memory configurations fail to deliver enough memory references per cycle to feed the SIMD functional units. This paper addresses the problem of the memory bandwidth.

We propose a novel mechanism suitable for 2-dimensional vector architectures and targeted at providing high effective bandwidth for SIMD memory instructions. The basis of this mechanism is the extension of the scope of vectorization at the memory level, so that 3-dimensional memory patterns can be fetched into a second-level register file.

By fetching long blocks of data and by reusing 2-dimensional memory streams at this second-level register file, we obtain a significant increase in the effective memory bandwidth. As side benefits, the new 3-dimensional load instructions provide a high robustness to memory latency and a significant reduction of the cache activity, thus reducing power and energy requirements. At the investment of a 50% more area than a regular SIMD register file, we have measured and average speed-up of 13% and the potential for power savings in the L2 cache of a 30%.

1 Introduction

Multimedia applications have become one of the most important types of workloads in current microprocessor design [1]. Most new general purpose and embedded processors include SIMD ISA extensions to increase the performance of future media protocols and killer applications such as MPEG-4 [2]. These new instruction extensions focus on exploiting data-level parallelism over small data-types (thus sometimes called μ-SIMD parallelism) inside a single register (64-128 bits typically). Examples of these new ISA extensions are INTELís MMX [3] and SSE[4], SUNís VIS[5], AMDís 3DNow! [6], MIPSís MDMX [7] and Motorolaís AltiVec [8].

Ranganathan et.al. [9] presented an in-depth study of the characteristics of μ-SIMD enhanced applications. They showed that after including software prefetching, most media applications were compute bound. Performance was, then, ultimately limited by fetch and issue bandwidth. In order to address this problem, several authors have proposed 2-dimensional vector architectures [10, 11, 12]. These architectures adapt to typical multimedia memory patterns by extending the scope of vectorization to two dimensions (or parallel loops). The main advantage of these 2-dimensional vector architectures is that they are able to signiÝcantly increase the number of operations per instruction, thus, breaking the fetch/issue barrier of most media programs.

In this paper, we study the behavior of several media applications using one of these 2D media extensions. We will show that several applications experience a signiÝcant performance degradation due to the the memory system. While data caches show an extremely high hit rate (as already highlighted by Slingerland et.al. [13]), they are, however, unable to deliver enough memory bandwidth for the vector functional units.

The design of high bandwidth cache memory systems is not trivial due to the complex memory layouts typically found in most media applications. In order to address this problem, we came to the observation that high amounts of spatial and temporal locality exist at extra dimensions of the memory pattern layout, even though there are computational dependences that do not allow straight-forward vectorization. This locality, if properly exploited, may enable high memory bandwidth with a feasible cache hierarchy based on widening the cache memory ports.

We propose a new extension to a 2D vector architecture targeted at implementing high bandwidth vector memory systems. The basis of this mechanism is a second-level vector register Ýle where 3-dimensional memory patterns can be fetched from the memory thanks to a new 3D vector load instruction. By doing this, we take advantage of higher amounts of spatial and temporal locality that translate into higher effective bandwidth and register reuse.

*This work has been supported by the Ministry of Science and Technology of Spain under contract TIC-2001-0995 and by the CEPBA

Proceedings of the 35 th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-35)
1072-4451/02 $17.00 © 2002 IEEE

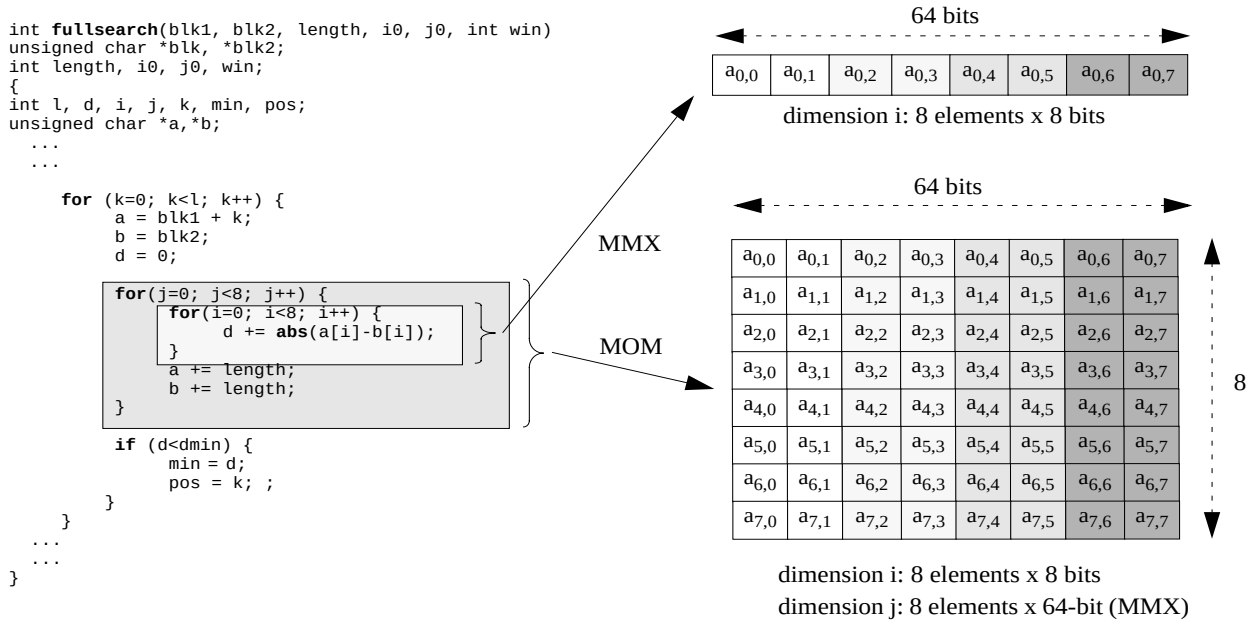


Figure 1. Comparison between (a) a conventional MMX-like μ-SIMD instruction and (b) a Matrix (MOM-like) 2D SIMD instruction.

We will show that our proposed mechanism is able to provide high performance gains for those applications where memory bandwidth is the main bottleneck. Even for the rest of the benchmarks, our proposal provides two significant side beneÝts: a sensible reduction of the cache activity and a prefetching effect. The former translates into lower power/energy consumption in the memory sub-system while the latter provides high robustness to the latency when the memory is far away.

2 A brief overview of a 2D vector ISA

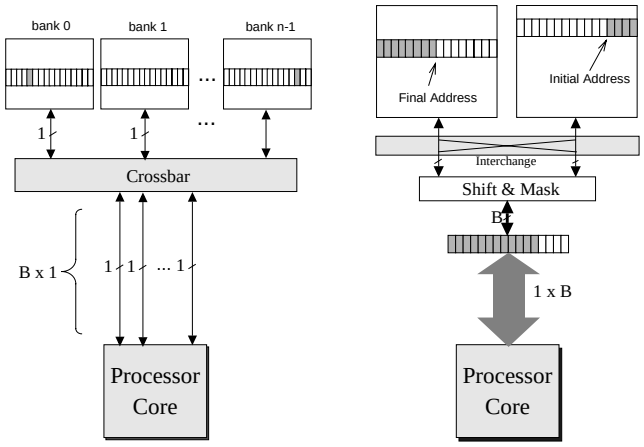
In this paper, we are going to use MOM [10] as our baseline 2D vector ISA. MOM stands for *Matrix Oriented Multimedia* extension and is a hybrid between a traditional vector and a μ-SIMD ISA. MOM is able to exploit up to two different dimensions of parallelism by using a different paradigm (either vector or μ-SIMD) to vectorize one of two available parallel nested loops.

MOM can be viewed as a conventional vector ISA where each of its computation operations are μ-SIMD MMX-like instructions. The execution of a MOM instruction is dictated by two different parameters. The *Vector Length* determines how many 64-bit elements of the MOM register are operated (out of 16). The *Vector Stride* determines the distance between two consecutive MOM vector elements when performing memory operations.

In order to help understand the differences between a conventional μ-SIMD approach and a 2D approach such as MOM, Ýgure 1 shows a simpliÝed fragment of code extracted from a MPEG-2 encoder. The algorithm shown is doing the *motion estimation* stage of the encoding, which detects movement of objects along different video frames. In order to do so, it searches across the reference image for the image block which matches better with the block being compressed. This is accomplished by Ýnding the minimal *sum of absolute differences* between the pixels of the two blocks. This search is performed, in the code, over several matrices laid out on the image x-axis. Note that `length` may be arbitrarily long, as it stands for the horizontal size of the frame.

Analyzing the code shown in the Ýgure we can see that there are up to two different dimensions of data-level parallelism to be exploited: nested loops i and j. The calculation of the sum of absolute differences between pairs of pixels (i, j) can be done in parallel fairly easily. Note, however, that loop k does not show the same property, as we have data and control dependencies in the if clause (clause that determines if we have found a local minimum) which avoid vectorization.

As shown in Ýgure 1, MOM is able to take advantage of the parallelism implicit in both loops i and j. First, it generates a MMX-like instruction for loop i, and then extends an additional vectorization of this instruction, replicating it across loop j. As a result, each pattern a and b are loaded into a single MOM register. In other words, each MOM register element (a 64-bit μ-SIMD register) corresponds to a row of a matrix.



(a) (b)

Figure 2. Cache designs for SIMD memory ports: (a) Multi-banking, (b) Port Widening.

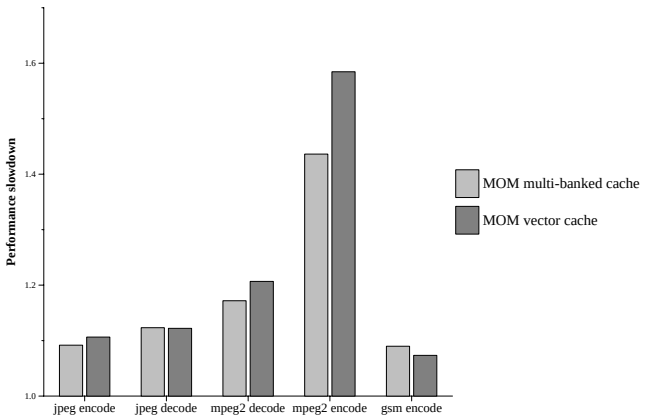


Figure 3. Performance slowdown for realistic memory system configurations.

3 Rationale for 3-dimensional vectorization

In this section we will show that 2D SIMD media programs can experience severe performance degradations due to the bandwidth constraints of realistic cache implementations. In order to address the problem, we will introduce two new instructions to perform 3D memory accesses and will discuss why they allow exploiting a higher amount of temporal and spatial locality.

3.1 The problem of the bandwidth

A traditional problem of SIMD architectures is the design of a memory system able to provide enough memory references per cycle to keep the SIMD functional units busy. As shown by Toni Juan et. al. [14], true multi-ported caches are not feasible due to their high cost. Alternative cache designs to true multi-ported caches are several, each with its drawbacks: time-multiplexing (as in the Alpha 21264 [15]), multi-banking, port widening, etc.

Multi-banking consists of implementing B memory ports connected with a set of cache memory banks by means of a crossbar (see Ýgure 2-a). A vector memory instruction can distribute its different memory references among all available memory ports. While this conÝguration presents the advantage of performing well for different strides, scalability is compromised because of bank contention and implementation issues of the crossbar for an elevated number of memory ports.

Port widening is a more restrictive (but cheaper) alternative, based on increasing the granularity of the memory accesses. Given a vector memory instruction whose elements are consecutively arranged in memory, we can fetch several elements in a single access provided that they are located in the same cache line. The vector cache [16] is a straightforward implementation of this concept. As shown in Ýgure 2-b, the vector cache is based on loading two whole cache lines (one per interleaved bank) instead of individually loading each vector element. Additional logic (an interchange switch, a shifter and a mask logic) allows selecting a chunk of up to B consecutive words, being the upper bound of B the size of a single cache line. Its main drawbacks are: Ýrst, it may add extra latency due to the shift&mask logic, and second, it is not able to provide more than one reference per cycle when the vector stride in different than one.

In order to evaluate the efÝciency of the two different cache designs, we have measured the performance degradation of a 8-way issue processor able to execute MOM instructions, for a set of benchmarks from *Mediabench* [17]. Figure 3 shows the processor performance slowdown for two different cache designs: (a) a 4-port multi-banked cache (with 8 memory banks), and (b) a vector cache with one single port of width 4×64 bits. Performance degradation is given relative to performance of an idealistic memory system (perfect cache, 1-cycle of latency, unbounded bandwidth). Details about the architecture conÝguration can be found in section 5.3.

Results show that some of the benchmarks have significant performance degradations when taking into account a realistic memory implementation (ranging from 8% to 58%). As the cache hit rates are relatively high (from 90% to 99%), the reason that explains such decreases in performance is no other than the effective bandwidth provided by the memory ports. Results also show that the vector cache obtains slowdowns reasonably similar to those of the multi-banked conÝguration, while being much easier to implement.

3.2 Identifying the potential of a third dimension

As seen in the previous subsection, some media benchmarks have severe performance shortcomings due to the

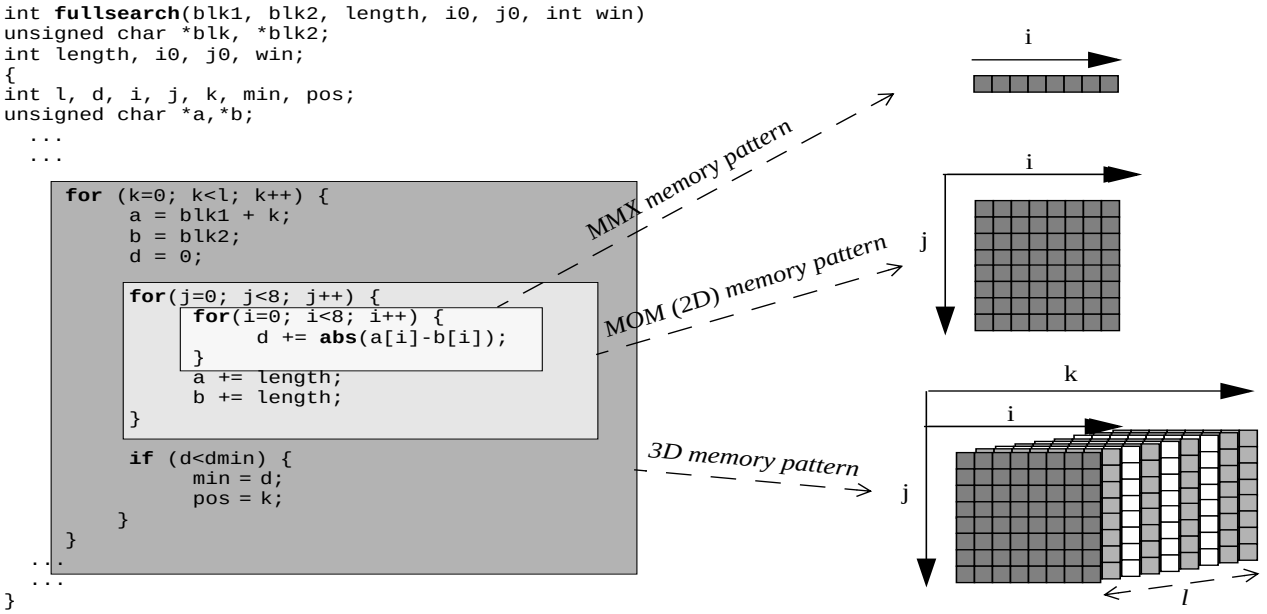


Figure 4. N-dimensional memory patterns in a MPEG2 kernel.

inability of any of the proposed vector memory systems to provide the required bandwidth. A way to identify the sources of the problem may come from a closer observation of MOM 2-dimensional memory pattern characteristics.

If we turn back at the example shown if Ýgure 1, we realize that there is a long distance between consecutive MOM elements (as the stride between two different MOM register elements corresponds to the horizontal size of the image). Therefore, a vector memory system such as the vector cache is unable to fetch more than one MOM register element per cycle, as two consecutive elements are placed in non-consecutive cache lines

Indeed, as already shown in [10, 11], strided matrices are a very common data structure in multimedia. These matrices are laid out in memory in such a way that, while the elements in a single row of one matrix are consecutively arranged in memory, elements beyond the Ýrst dimension are distributed across far away cache lines. From the set of benchmarks, only `jpeg decode` and `gsm encode` own memory patterns characterized for wide blocks of consecutive data along a single dimension. To solve this problem, some authors propose simply rearranging the data to Ýt a better layout. We have found that most of the times is either not possible (due to the way the benchmarks are written) or counterproductive (since it may produce even worse memory behavior in other stages of the applications).

Our claim is that the solution for this problem may reside in the exploitation of more dimensions of the media memory layout than those already exploited by 2D vectors ISAs. More dimensions bring more opportunities to Ýnd longer sets of data consecutively arranged in memory, and hence, more opportunities to fully exploit the peak bandwidth of a wider memory port.

If we look further into the n-dimensional structure of media data, we can realize that a set of MOM 2-dimensional streams as a whole shows a higher level of spatial and temporal locality than every stream in isolation. If we reorder the way we access the streams, we can take advantage of the existence of longer chunks of data and from the redundancy intrinsic to the overlapping of different 2-dimensional streams.

In the previous example (see Ýgure 4), the row elements of matrices a and b are extremely sparse. Therefore, if we use a vector cache, we are only able to gather the eight 8-bit elements of one row with a single access.

Nevertheless, when looking at the third dimension of the algorithm (corresponding to loop k), we can observe a 3-dimensional memory pattern composed of a set of 2-dimensional matrices. These 2D matrices are laid out on the x-axis of the image (the loop i) with an address offset (or stride) of one single byte. The overall structure is a rectangular matrix of length $(8 + l - 1) \times 8$. The interesting point of this structure is that it exposes several elements consecutively arranged in memory and that it determines a high amount of potential MOM 2D memory streams inside, as there is a high amount of overlapping between them.

The main point is that, even though k loop cannot be fully vectorized, we can vectorize the memory access to this 3-dimensional memory pattern, as there are no memory dependences between the matrices of every instance of loop k.

By doing so, we are able to increase the effective memory bandwidth (as we are exposing longer chunks of data), and we are able to reduce the memory trafÝc (as we can avoid fetching repeatedly redundant data when streams overlap).

4 3D memory vectorization

We propose a novel vector memory access technique based on implementing a new set of *3D vector registers*. These 3D vector registers will be used as temporal storage for 3D memory streams fetched from memory. By doing sequential accesses to these second-level registers, we will be able to conveniently rearrange the 3D memory pattern to accommodate 2D MOM memory accesses.

A 3D vector register is basically a widened version of a common MOM register. A *3D vector load* instruction allows to transfer multiple cache lines inside the different elements of a single 3D vector register. Afterwards, the data in the 3D register Ýle can be transferred to the MOM register Ýle using a *3D vector move* instruction. In the same vein that the MOM register Ýle, the 3D register Ýle is organized in lanes (or clusters). This organization enables very high bandwidth transfers with low hardware complexity.

It is very important to note that from a compiler/programmer point of view, 3D memory instructions can be used even if the third outer loop is not strictly vectorizable. We are using these instructions to strictly fetch data from memory and to rearrange the data later on. Therefore, those computational dependences not related to read/write conÐicts between the 2D memory streams can be ignored.

Our proposed 3-dimensional memory vectorization technique provides three signiÝcant advantages:

- longer chunks of data accessed every cycle
- reduction of cache trafÝc by means of register reuse
- more elements packed per vector memory instruction

In this paper, we will quantify how well the 3D memory instructions do improve the length of the chunks of data to be accessed, reduce the cache trafÝc and increase the number of elements packed per memory instruction. Finally, we will evaluate the impact of these factor over performance, power and robustness to the latency.

4.1 Semantics of the 3D memory instructions

We have used the MOM Instruction Set Architecture [18] as a representative example of a 2D media vector ISA. Our objective is to evaluate the potential of extending a 2D instruction repertoire with 3D memory instructions. The *MOM* Instruction Set Architecture contains 121 instructions and 16 logical 2D vector registers. Each 2D vector register is composed of 16 MMX-like elements of 64-bit each. The ISA includes a *Vector Length register* that keeps track of the number of MOM elements to be operated. Additionally, MOM memory instructions include an extra Ýeld containing the *Vector Stride* to control the load and store of 2D memory patterns.

We have made two modiÝcations to the basic MOM architecture: the set of logical registers has been expanded with the inclusion of two *3D vector registers*, and the instruction repertoire includes two new instructions designed to transfer data to/from these new registers.

A *3D vector register* is a widened version of a regular MOM register (see Ýgure 5 for a comparison of both kinds of registers). Instead of 16 elements of 8 bytes, a 3D vector register contains 16 elements of 128 bytes (16 x 64 bits), enough to Ýt a typical L2 cache line. Every 3D vector register has also a 7-bit *pointer* register, which maintains the current offset within the 3D vector register. This offset determines which slice of data is going to be transferred to a 2D MOM vector register.

The two new instructions have the following syntax and semantics:

3D Vector Load. This instruction has the form `Dvload` $3DR_i$ `<=` R_j`,` R_k`, W, b.` $3DR_i$ is one of the two 3D logical vector registers. R_j is the base address where the load starts. R_k is the *vector stride*. W is an 4-bit immediate value which indicates the width of each 3D-register element. Finally, b is a Ðag that indicates the initial value of the 3D-register pointer.

The semantics of the instruction are as follows (see Ýgure 5-a): starting at address R_j, load a block of $W \times 64-bit$ into the Ýrst position of 3D register i. Repeat the process, adding the stride register R_k to the current base address, for the next $VL-1$ elements of the 3D register (being VL the contents of the *Vector Length* register). The value of the register pointer is either the beginning or the end of the register, according to the value of the Ðag b (this allows to move along the two ways of the third dimension).

3D Vector Move. This instruction allows to move one subset of the 3D logical vector register into a 2D MOM register and has the form `3dvmov` MR_i `<=` $3DR_j$`,` P_s`.` MR_i stands for the MOM destination register. $3DR_j$ is the 3D logical vector register from where the data is going to be transferred. P_s is the pointer stride.

The semantics of the instruction are as follows (see Ýgure 5-b): starting at $offset$ ($offset$ being the contents, in bytes, of the *pointer register* associated with $3DR_j$), move a 64-bit sub-block from 3D-register j to the MOM register Ýle i. This process is repeated VL times (VL being the contents of the *Vector Length* register). Finally, update the current value of the 3D register pointer by adding P_s.

5 Evaluation background

In this section we present the methodology we have followed to evaluate the beneÝts of the 3D memory vector extensions to the MOM ISA, and we quantify the improve-

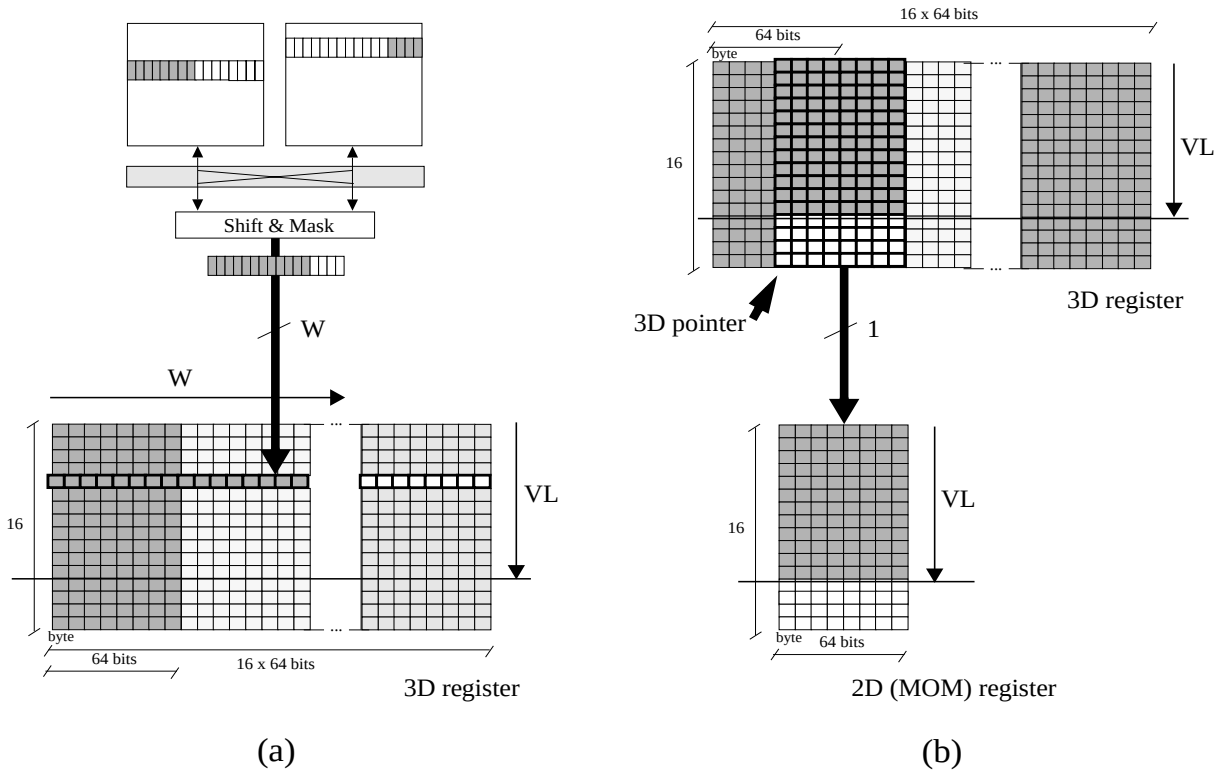


Figure 5. 3D vector memory instructions (a) 3D vector load (from the vector cache to one 3D register) (b) 3D vector move (from one 3D register to one MOM register).

ments of the new 3D memory instructions compared with the original 2D memory instructions.

5.1 Benchmarks and Code Generation

We have used the set of modiÝed benchmarks described in [10]. From the *Mediabench* suite [17], the authors rewrote a set of representative examples of video, image and audio applications, using two versions of media ISA extensions: a 1D μ-SIMD ISA (similar to MMX) and MOM. We have selected those with the highest vectorization percentage: `mpeg2 encode`, `mpeg2 decode`, `jpeg encode`, `jpeg decode` and `gsm encode`. The benchmarks show a wide selection of types of media memory streams, thus being suitable for evaluating the generality of our 3D memory instructions.

We have modiÝed the emulation libraries and traces obtained using ATOM [19], so that we are able to include 3D memory instructions to the MOM versions of the benchmarks. The 3D memory instructions were added to those loops that fulÝlled either of the following conditions: (a) there was potential to fetch more that one MOM stream by loading a whole cache line, and (b) there was potential for reuse at the 3D register Ýle level due to overlapping between two or more MOM memory streams. From the set of benchmarks, only `jpeg decode` did not have suitable 3-dimensional memory patterns to be exploited with our technique.

For our initial evaluation, the 3D enhanced code has been hand-written after a careful study of the algorithms. We believe, however, that the compiler support needed for generating such instructions is relatively feasible to implement, due to the nature of the analysis. Since we are only vectorizing memory references, we do not need to check dependences beyond those related to conÐicting reading and writing 2D memory streams. As media kernels usually have lots of 2D loads and no 2D stores, the analysis is commonly trivial (detecting the stride between the 2D load instructions to pack them together into a single 3D load and replacing the original 2D load instructions with 3D vector moves).

5.2 Characteristics of the new instructions

In the previous section, we claimed that the performance beneÝts from the new 3D memory instructions would come from three main factors. In this section we will brieÐy quantify them and discuss their beneÝcial impact over the architecture.

A. Longer data chunks accessed per cycle. Our 3D memory instructions focus on fetching wider blocks of data to capture slices from different MOM memory streams. As a result, they exhibit the potential to obtain more effective

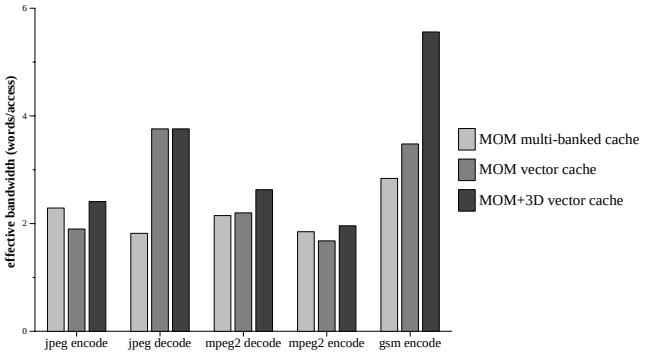


Figure 6. Effective memory bandwidth (in words transferred per access) for the different memory systems and ISA enhancements.

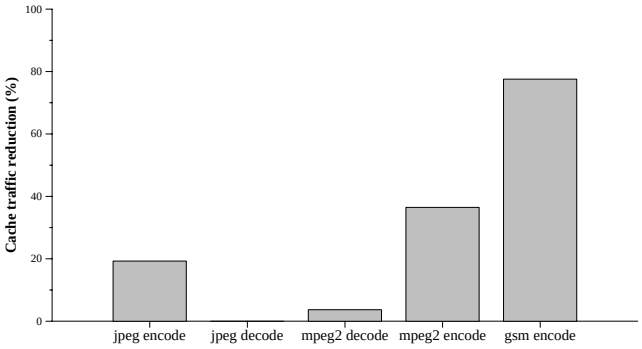


Figure 7. Vector cache traffic reduction when using 3D vectorization (in 64-bit words transferred).

bandwidth from a vector cache conÝguration, that is able to access as many elements as the width of a cache line. To show this property, Ýgure 6 shows the effective bandwidth of different cache implementations with and without 3D instructions. We consider the effective bandwidth to be the average number of words that can be obtained with a single access to the cache (or to several banks concurrently in the case of the multi-banked cache).

As shown in the Ýgure, 3D memory vectorization makes very good use of the simple vector cache implementation, increasing the effective memory bandwidth for several benchmarks and being even better than the expensive multi-banked conÝguration.

Having longer consecutive sets of data to access each cycle will translate into two main beneÝts. First, we will increase the effective bandwidth of the vector memory system, thus reducing performance slowdown. Second, we will gather more data every time we access the cache, thus reducing the cache activity (and as a direct consequence, the power consumption).

B. Reduction of the cache traffic. As we have a second-level register Ýle that is aware of the behavior of the memory references at the third dimension, we have opportunities to reduce the trafÝc to the cache by means of reusing (totally or partially) streams at the 3D register Ýle level. For instance, we may have 2D streams with data overlapping (as in the example of section 2), or sets of 2D streams that become invariant at the third dimension of the nested loops.

In order to realize the impact of register reuse over trafÝc reduction, we may look at Ýgure 7. In the Ýgure, we present the vector cache trafÝc reduction when including a 3D vector register Ýle, measured as the reduction of 64-bit words transferred from or to the vector cache sub-system.

Reusing data at the register Ýle level has a clear impact on the power consumption of the system (as the accesses to the 3D register Ýle are cheaper, in energy terms, than the accesses to the cache banks). Additionally, the latency of the 3D register Ýle is much shorter than the cache, thus providing a way to alleviate the processor-memory speed gap impact.

	MOM			MOM + 3D		
	1st	2nd	3rd	1st	2nd	3rd (max)
mpeg2encode	7.2	10.1	ñ	7.2	9.3	1.5 (5)
mpeg2decode	4.2	7.4	ñ	4.2	6.2	1.7 (3)
jpeg encode	4.1	8.2	ñ	4.1	7.8	1.9 (16)
jpeg decode	5.5	15.9	ñ	5.5	15.9	ñ
gsm	4.0	10.0	ñ	4.0	10.0	7.7 (16)

Table 1. Memory instruction vector length for each of the three dimensions.

C. Longer vector memory instructions. It is widely known that the longer the vectors of a given architecture, the better the ability to tolerate memory latency. Our 3D memory architecture extension provides two main beneÝts that have the potential to better tolerate increases in the latency of the memory instructions. First, we are actually doing a sort of software prefetching, as a 3D memory instruction triggers the fetching of streams of data several cycles before they will be really needed. Second, we pack more elements per memory instruction, thus taking advantage of the relation between the vector length and the tolerance to memory latency.

Table 1 presents the average vector length along each dimension in every memory instruction (two dimensions for plain MOM memory instructions, three dimensions when including 3D memory instructions). Taking into account that the 3D memory instructions are typically less predominant than the 2D memory instructions (as they are around 4 times longer, and hence, fewer 3D loads are required when taking advantage of the 3D register reuse), we may realize how the third dimension is contributing to the amount of data read by each instruction.

	MMX	MOM
Fetch rate	8	8
graduation window	128	128
Load/Store queue	32	32
INTEGER issue	4	4
INTEGER FUs	4	4
SIMD issue	4	1
SIMD FUs	4	1x4
memory issue	4	2
L1 memory ports	4	2
L2 vector memory ports	n/a	1x4

Table 2. Processor configurations.

5.3 Modeled architecture

We have used the *Jinks* simulator [10] to model an aggressive 8-way out-of-order superscalar processor. The processor is enhanced with its own independent multimedia pipeline and SIMD register Ýle. We have two versions of the same model, able to execute either MMX-style or MOM instructions. Architectural parameters are summarized in table 2. As seen, the MMX conÝguration is aggressive in number of registers and functional units to avoid an unfair comparison with MOM.

Note that the MOM processor has one SIMD functional unit with four lanes or clusters. Every cluster is able to perform one MOM operation/cycle from the same MOM instruction, thus providing overall the same FU bandwidth than the MMX processor (Ýgure 8-b illustrates the MOM lane conÝguration).

In order to implement the combined 2D/3D memory mechanism in the MOM architecture, we need to include two new register Ýles: the*3D Vector Register File*, that contains 4 physical 3D vector registers and the *3D Pointer Register File* which keeps the coherent values of the pointers for each logical 3D vector register. Note that the renaming process of the 3D physical vector registers and the physical pointer registers is not the same. For instance, a `3dvmov` operation (which moves a slice from a 3D vector register to a MOM register) causes the pointer register to be renamed, as its value is updated using the pointer stride. Table 3 summarizes the different register Ýle conÝgurations. We have assumed 3 cycles of latency for the 3D vector register Ýle (but 1 cycle per transfer).

We have estimated the area cost of the different register Ýles using the models described in [20]. Estimated register Ýle areas (in square wire tracks) and overall normalized areas (relative to the MMX-like processor) are included in table 3.

Figure 8 shows the vector memory sub-system implementation. Our basic cache hierarchy model is similar to the Alpha 21364 [21] one, where both L1 and L2 caches are located on-chip. The L1 cache is a 64 KB, 2-way set associative, write-through cache with 32-byte lines. The L2 cache is a 2MB, 4-way set associative, write-back cache

	MMX	MOM	MOM + 3D
	MMX/MOM Register File		
register size	64 b	16x64b	16x64b
logical/physical registers	32/80	16/36	16/36
read ports (per lane)	12	3	3
write ports (per lane)	8	2	2
max memory bandwidth	4	4	4
estimated area (wt^2)	2,826,240	2,654,208	2,654,208
cache buses (wt^2)	262,144	262,144	n/a
	Accumulator Register File		
register size	n/a	192b	192b
logical/physical registers	n/a	2/4	2/4
read ports	n/a	1	1
write ports	n/a	1	1
estimated area (wt^2)	n/a	23,040	23,040
	3D Vector Register File		
register size	n/a	n/a	16x16x64b
logical/physical registers	n/a	n/a	2/4
read ports (per lane)	n/a	n/a	1
write ports (per lane)	n/a	n/a	1
max memory bandwidth	n/a	n/a	16
estimated area (wt^2)	n/a	n/a	1,966,080
	3D Pointer Register File		
register size	n/a	n/a	7b
logical/physical registers	n/a	n/a	2/8
read ports	n/a	n/a	2
write ports	n/a	n/a	2
estimated area (wt^2)	n/a	n/a	3,136
Estimated RF area	3,088,384	2,939,392	4,646,464
Overall normalized area	1.00	0.95	1.50

Table 3. Multimedia register file configurations.

with 128-byte lines. L1 data cache latency is 1 cycle while L2 cache latency is 20 cycles. The instruction cache has not been simulated given the extremely low instruction miss rates measured.

We have decided to adopt the same cache hierarchy conÝguration proposed for the original MOM architecture [16, 22]. In this architecture, the MOM memory accesses bypass the L1 cache and go straight to the L2 cache. As interference between vector and scalar data might occur, a simple-coherence protocol, based on an exclusive-bit policy, was proposed.

Several reasons explain why is worth paying the extra latency and implementing the vector memory sub-system over the second level of cache. First, we avoid jeopardizing the L1 cycle time and latency, thus not compromising scalar performance, which is paramount for the target architecture. Second, the L2 cache has longer cache lines than the L1 data cache, hence increasing the potential performance of the already cost-efÝcient vector cache implementation.

Looking at Ýgure 8-a and 8-b , we can compare the implementation of a 4-port multi-banked cache and a vector cache for the original MOM architecture. The interconnection logic of the vector cache is signiÝcantly simpler than its multi-banked counterpart. Note, however, that the vector cache peak bandwidth is limited by the number of lanes of the MOM pipeline (4 for our conÝguration). Even though

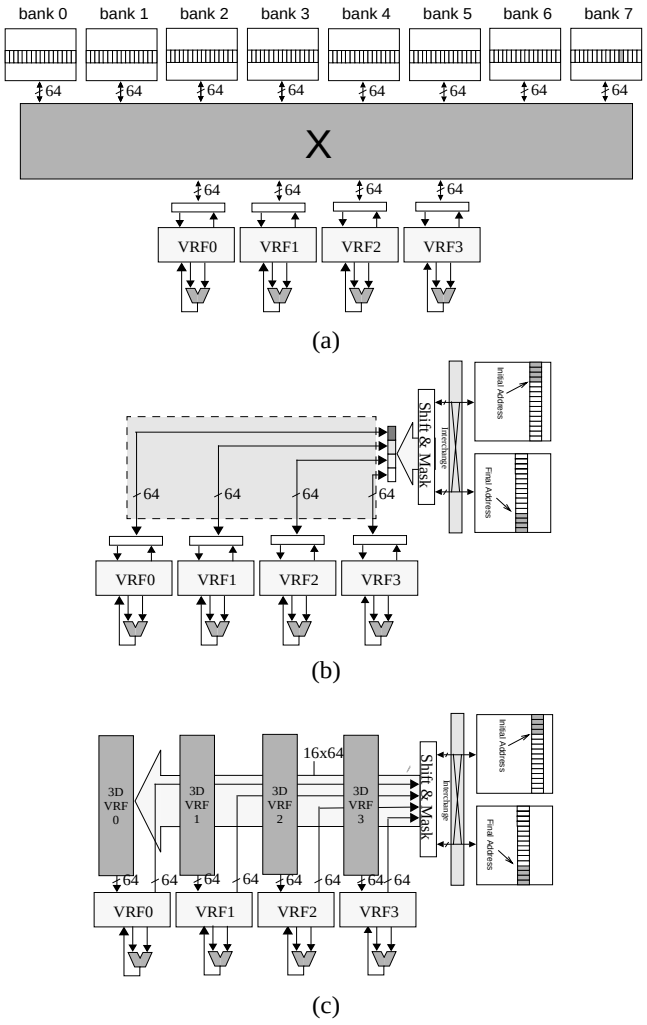


Figure 8. Vector memory sub-system implementations: (a) multi-banked cache, (b) vector cache, and (c) vector cache and second-level 3D vector register file.

the 4x8 crossbar required for the multi-banked cache is not simple, we have not considered any extra latency to the cache access pipeline.

Figure 8-c shows the vector memory system implementation, but this time for the MOM architecture with 3D memory instructions. Note that the 3D vector register Ýle is distributed over as many lanes as the MOM register Ýle. The different widened elements of the 3D physical vector registers are distributed within these lanes. All the different 3D vector lanes are connected to the same array of bitlines. So, every cycle, a chunk of up to 128 bytes of data can be fetched from the L2 cache and can be directly written in parallel to *one* of the 3D vector register Ýle lanes. Therefore, the effective memory bandwidth may be as large as the size

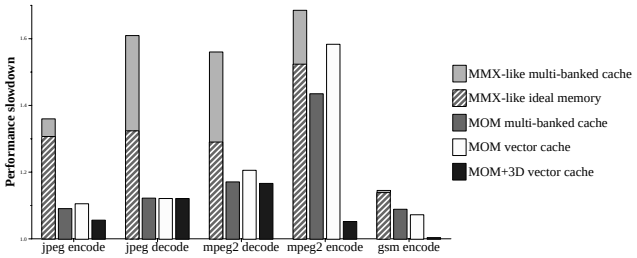


Figure 9. Performance slowdown for the different ISA and memory sub-system configurations.

of a whole L2 cache line. While large chunks of data are written in one of the 3D vector lanes, one 64-bit element can be read from each of these lanes. As a result, we have a peak transfer rate of four 64 bits elements per cycle between the 3D vector register Ýle and the MOM register Ýle. Note that the 3D register Ýle allows byte-aligned accesses. From the point of view of implementation, we would typically require a mechanism that fetches two consecutive quadword-aligned elements and that is able to use a shift&mask logic block to extract the required 64-bit element.

6 Performance and power benefits of 3D memory vectorization

In this section we will evaluate the beneÝts provided by 3D memory vectorization in terms of performance slowdown relative to an idealistic memory system and will analyze the impact of increasing the cache latency. Finally, we will roughly estimate the power savings leveraged by the reduction of the cache activity.

6.1 Performance slowdown with realistic memory

Figure 9 shows the performance slowdown of different ISA and memory sub-system conÝgurations, relative to the performance of a MOM processor with an idealistic memory system (single cycle of latency, effective bandwidth equal to the peak bandwidth). The Ýgure allows us to determine how well a given memory system performs over a speciÝc ISA style.

First, Ýgure 9 allows us to see the effect of a realistic memory implementation over the performance of the MMX-like conÝguration processor. As seen in the Ýgure, software prefetching combined with a way of avoiding bank collisions would approximate the performance of the MMX-like system to the one of an idealistic memory system, but it would still be far from the performance of the idealistic MOM system (1.31X of performance slowdown in average). The reason is that the MMX-style processor is limited by issue bandwidth and not by memory bandwidth.

On the other hand, the realistic memory conÝgurations of the MOM processor do not behave much better than their MMX-style idealistic counterpart. The vector cache presents performance slowdowns ranging from 1.07X to 1.58X (1.22X in average), while the much more expensive multi-banked cache improves very slightly those results, with slowdowns ranging from 1.09X to 1.52X (1.19X in average).

While 3D memory vectorization is not the panacea for all media benchmarks (as some of them are clearly not limited by memory bandwidth), results show that is an ideal candidate for solving the memory problem of the most critical ones. Using the simple vector cache implementation as a basic core, the 3D memory vectorization achieves high performance increases in those benchmarks where the memory impact is higher (such as in `mpeg2encode`, where performance is improved by a 55%). Overall, performance slowdown ranges only from 1.005X to 1.16X (1.08X in average), clearly demonstrating the capability to overcome the memory barrier of our proposed technique.

6.2 Robustness to increases in the memory latency

As mentioned previously, our 3D memory vectorization mechanism resembles a software prefetching technique in certain aspects. We are able to tolerate high increases in the memory latency since: (a) we are fetching to the 3D register Ýle several 2D memory streams before they are really needed, (b) we have longer memory streams and (c) we are reusing data at the 3D register Ýle, which is considerably faster than the memory.

We have evaluated the performance of the architecture with and without 3D memory extensions in the scenario of 40-60 cycles of L2 cache latency. Such an experiment is interesting, given the current technology trends, where L2 latency is bound to rise due to the increasing predominance of wire delays. Moreover, this experiment allows us to extend the scope of our architecture to in-memory processors such as VIRAM [23], where DRAM main memory is located on-chip and no SRAM L2 cache is implemented.

Figure 10 shows normalized execution time for MOM and MOM with 3D memory extensions, when we increase the L2 cache latency. Results show that the 3D memory architecture is much more latency tolerant than its basic MOM counterpart. MOM average slowdown is 1.27X when increasing L2 latency from 20 cycles to 40, while MOM + 3D memory extensions slowdown is only 1.18X. At 60 cycles, relative speed-up between MOM and MOM + 3D memory extensions rises to 11% for `jpeg encode`, 10% for `mpeg2decode` and 16% for `gsm encode`.

6.3 Power Estimations

The use of 3D memory vectorization may have interesting potential from the point of view of the power consumption of the memory system. As already seen in previous sections, 3D memory vectorization reduces the activity of the cache subsystem by: (a) increasing the number of elements fetched per access, and (b) reducing the overall trafÝc via register reuse.

	multi-banked	vector cache	vector cache + 3D reg. Ýle
jpeg encode	6.30	4.23	2.53
jpeg decode	3.82	2.46	2.46
mpeg2 decode	3.39	2.59	2.08
mpeg2 encode	39.88	38.48	21.00
gsm encode	6.21	2.31	0.32

Table 4. L2 cache activity (in Millions of accesses to L2).

Table 4 shows the L2 cache activity for the three different memory sub-system implementations (multi-banked, vector cache and vector cache with a 3D register Ýle). Results show that the multi-banked cache, being an already high cost implementation, consumes much more energy than the vector cache implementation, as it does not take beneÝt from fetching more than one data element from the same cache line every cycle. As a result, the vector cache conÝguration reduces an average of 31% in the number of overall accesses (relative to its multi-banked counterpart). On the other hand, introducing 3D memory vectorization reduces activity an additional 38% (relative to the raw vector cache).

While cache activity is a good measurement for evaluating potential power savings, the cost of accessing the 3D register Ýle could offset the overall power consumption. In order to present a rough estimation of potential, we have used the power models described by Rixner et.al. [20] to evaluate the power consumption of the L2 cache and the 3D register Ýle for the multi-banked conÝguration, the vector cache conÝguration, and the vector cache plus 3D register Ýle conÝguration (assuming a 0.18μm CMOS 1 GHz processor). The model is an approximation, as some optimizations (such as the use of hierarchical or differential bit lines) have not been considered. We have assumed that the L2 cache is physically distributed across 32 memory sub-arrays

Figure 11 shows average power consumption in watts of the L2 cache combined with the 3D register Ýle (for our 3D enhanced architecture). Result shows that our 3D vector register Ýle consumes a negligible amount of power compared to the savings we obtain in the L2 cache. As a result, our 3D enhanced processor appears as an energy-efÝcient architecture, as we are reducing the execution time by 13% in average while reducing at the same time the power consumption of the L2 cache by 30%.

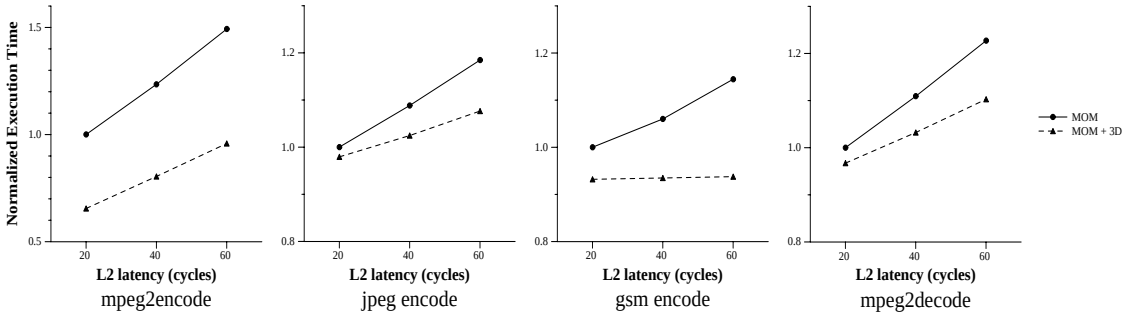


Figure 10. Normalized execution time for different L2 cache latencies with and without 3D memory instructions.

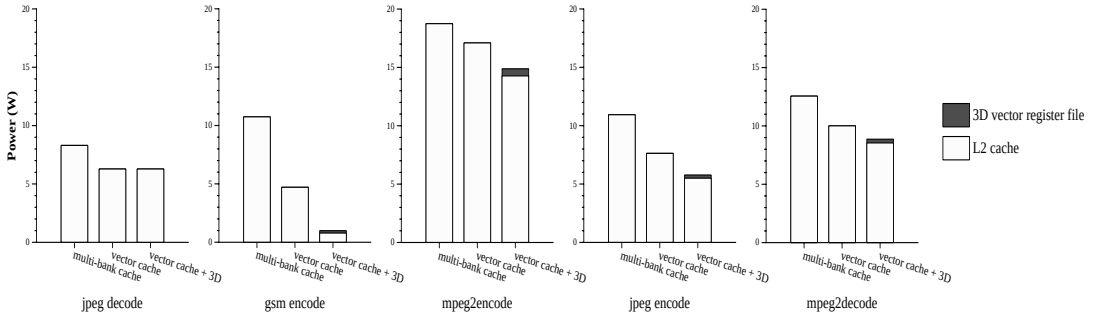


Figure 11. Memory sub-system (L2 cache + 3D RF) average power consumption for the different configurations.

7 Related Work

There are several vector/SIMD architectures that could beneÝt from the 3D memory vectorization philosophy presented in this paper, due to their utilization, up to a certain extent, of 2-dimensional memory patterns. Some examples are the VIRAM processor [23], the CSI architecture [11], the *Imagine* processor [24], or the PlayStationís *Emotion Engine* [25]. Even more general n-dimensional architectures such as *MediaBreeze* [26] could generalize the concept to implement feasible high-bandwidth memory ports.

In the DSP domain, there is extensive work dealing with n-dimensional prefetching or data reorganization for multimedia applications [27, 28]. Zhang et. al. [29] use the *Impulse* memory controller to gather sparse media streams into dense cache lines for a set of image processing applications. Additionally, the distribution of our 3D register Ýle in clusters to provide high bandwidth at low cost is similar to the *Rake cache* proposed by Asanovic [30], which is a distributed cache for each vector lane. Our proposal is focused on read-only memory streams and does not have the coherency issues of the Rake cache.

There have been several works related to the idea of second level vector register Ýles and vector features to increase the percentage of register reuse. The NEC SX-3 vector architecture [31] featured a second level vector register Ýle with longer register Ýles than the regular ones. That idea was to have a binding prefetch mechanism targeted to hide memory latency with longer vector lengths. From the point of view of vector register reuse, the CONVEX C4000 processor features the `vector first` instruction, which allows skewing a whole vector by a single position, and reuses them for the next iteration of the related vector instruction.

Even from the point of view of basic vector coding techniques, several tricks can be used to mimic, in a limited way, what 3D memory vectorization does. A combination of vector shift and mask instructions could be used to take advantage of overlapping 2D streams, but at the cost of a high instruction overhead, and an increase in pressure over the 2D register Ýle. Moreover, these kind of techniques take advantage of the overlap between streams but cannot take advantage of the fetching of wide blocks of cache in a single access, as our technique does.

8 Summary

This paper has shown that several media applications suffer severe performance degradations due to the memory bandwidth. We have seen that caches focused on fetching wide blocks of data are cost/effective but fails sometimes to

provide enough memory bandwidth due to the sparse behavior of common 2D multimedia memory patterns.

We have proposed a combined 2D/3D memory vectorization mechanism that can be adapted to a general 2D SIMD architecture such as MOM. The mechanism is based of fetching long 3D streams of data into a second level register Ýle, used only for load memory transfers. This 3D streams can be accessed by slices efÝciently with a simple clustered distribution of the register Ýle, providing high bandwidth to feed the 2D SIMD functional units.

This 3D memory vectorization mechanism leverages two main advantages. We increase the effective memory bandwidth by fetching wider blocks of data that will be conveniently reorganized, and we reduce the overall trafÝc by reusing data inside the 3D registers due to overlap between 2D streams.

Our mechanism, implemented on a 8-way superscalar processor with 2D vector extensions, reduces signiÝcantly the performance degradation of memory-bound benchmarks. Additionally, the associated reduction of the cache activity shows the potential of power savings at the level of the cache sub-system. Therefore, at the cost of a 50% more area than a regular MMX-style register Ýle, we provide a mechanism that leverages a 13% of performance speed-up and achieves a 30% of power savings in the L2 cache.

References

[1] K. Diefendorff and P.K. Dubey. How multimedia workloads will change processor design. *IEEE Micro*, pages 43ñ45, Sep 1997.

[2] Rob Koenen. Mpeg-4, multimedia for our time. *IEEE Spectrum*, pages 26ñ34, February 1999.

[3] A. Peleg and U. Weiser. Mmx technology extension to the intel architecture. *IEEE Micro*, pages 43ñ45, August 1996.

[4] Pentium iii processor: Developerís manual. Technical Report http://developer.intel.com/design/PentiumIII, INTEL, 1999.

[5] M. Tremblay, J.M. OíConnor, V. Narayanan, and L. He. Vis speeds new media processing. *IEEE Micro*, August 1996.

[6] 3dnow! technology manual. Technical Report http://www.amd.com, Advanced Micro Devices, Inc., 1999.

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

[8] K. Diefendorff, P.K. Dubey, R. Hochsprung, and H. Scales. Altivec extension to powerpc accelerates media processing. *IEEE Micro*, pages 85ñ95, March-April 2000.

[9] 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.

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

[11] Ben Juurlink, Dmitri Tcheressiz, Stamatis Vassiliadis, and Harry Wijshoff. Implementation and evaluation of the complex streamed instruction set. *Parallel Architectures and Compilation Techniques, PACT-01, Barcelona*, September 2001.

[12] I. Watson A. El-Mahdy. A two-dimensional vector architecture for multimedia. *EUROPAR*, 2001.

[13] N.T. Slingerland and A. J. Smith. Cache performance for multimedia applications. *International Conference on Supercomputing, ICS*, pages 204ñ217, Sorrento, Italy 2001.

[14] T. Juan, J. Navarro, and O. Temam. Data caches for superscalar processors. *International Conference on Supercomputing, ICS97*, pages 60ñ67, 1997.

[15] R.E. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, pages 24ñ36, March-April 1999.

[16] 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.

[17] C. Lee, M. Potkonjak, and W.H. Magione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communication systems. *MICRO 30*, 1997.

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

[19] A. Srivastava and A. Eulace. Atom: A system for building customized program analysis tools. *Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation*.

[20] S. Rixner, W.J. Dally, B. Khailany, P. Mattson, U. Kapasi, and J.D. Owens. Register organization for media processing. *High Performance Computer Architecture, HPCA-5*, pages 375ñ386, 2000.

[21] Peter Bannon. Alpha 21364: A Scalable Single-chip SMP. Technical Report http://www.digital.com/alphaoem/microprocessorforum.htm, Compaq Computer Corporation, 1998.

[22] Jesus Corbal, Roger Espasa, and Mateo Valero. Dlp + tlp processors for the next generation of media workloads. *HPCA*, January 2001.

[23] Christoforos Kozyrakis. A media-enhanced vector architecture for embedded memory systems. *Technical Report UCB//CSD-99-1059*, July 1999.

[24] William J. Dally. Tomorrowís computing engines (keynote speech). Feb 1998.

[25] A. Kunimatsu, N. Ide, and T. Sato et. al. Vector unit architecture for emotion synthesis. *IEEE Micro*, pages 85ñ95, March-April 2000.

[26] Deependra Talla and Lizy K. John. Cost-effective hardware acceleration of multimedia applications. *2001 IEEE International Conference on Computer Design (ICCD)*, September 2001.

[27] F.Catthoor, S.Wuytack, E.De Greef, F.Balasa, L.Nachtergaele, and A.Vandecappelle. Custom memory management methodology ñ exploration of memory organisation for embedded multimedia system design. *Kluwer Acad. Publ., Boston*, ISBN 0-7923-8288-9, 1998.

[28] R.Schaffer, F.Catthoor, and R.Merker. Combining background memory management and regular array co-partitioning illustrated on a full motion estimation kernel. *special issue on* Advanced Regular Array Design *(T.Plaks, ed.) in* J. of Parallel Algorithms and Applications, Vol.15, No.3-4:pp.201ñ228, December 2000.

[29] L. Zhang, J.B. Carter, W.C. Hsieh, and S.A. McKee. Memory system support for image processing. *the 1999 International Conference on Parallel Architectures and Compilation Techniques (PACT'99),*, pages pp. 98ñ107, October 1999.

[30] Krste Asanovic. Vector microprocessors. Phd thesis, University of California at Berkeley, 1998.

[31] Akihiro Iwaya and Tadashi Watanabe. The parallel processing feature of the NEC SX-3 supercomputer system. *Intl. Journal of High Speed Computing*, 3(3&4):187ñ197, 1991.