A PCI AER Co-Processor Evaluation Based on CPUs Performance Counters

Manuel Domínguez-Morales, Alejandro Linares-Barranco, Pablo Iñigo-Blasco, Juan Luis Font, Daniel Cascado-Caballero, Gabriel Jimenez-Moreno, Fernando Díaz-del-Río, José Luis Sevillano
Department of Architecture and Technology of Computers, University of Seville, Spain
mdominguez@atc.us.es

Abstract

Image processing in digital computer systems usually considers the visual information as a sequence of frames. These frames are from cameras that capture reality for a short period of time. They are renewed and transmitted at a rate of 25-30 frames per second, in a typical real-time scenario. Digital video processing has to process each frame in order to obtain a filter result or detect a feature on the input. This processing is usually based on very complex and expensive (in resources) operations for an efficient real-time application. Brain can perform very complex visual processing in real-time using relatively simple cells, called neurons, which codify the information into spikes. Spike-based processing is a relatively new approach that implements the processing by manipulating spikes one by one at the time they are transmitted, like a human brain. The spike-based philosophy for visual information processing based on the neuro-inspired Address Event Representation (AER) is achieving nowadays very high performances. In this work we study the low level performance for real-time scenarios of a spike-based co-processor connected to a conventional PC and implemented through a PCI board. These low level lacks are focused both in the software conversion of static frames into AER format and in the bottleneck of the PCI interface.

Keywords: Spiking neurons, Address-Event, PCI, FPGA, Synthetic AER generation.

1 Introduction

Digital vision systems process sequences of frames from conventional frame-based video sources, like cameras. For performing complex object recognition algorithms, sequences of computational operations must be performed for each frame. The computational power and speed required make it difficult to develop a real-time autonomous system. However, brain performs powerful and fast vision processing using millions of small and slow cells working in parallel in a totally different way. Primate brain is structured in neuron layers, in which the neurons in a layer connect to a very large number (~10^4) of neurons in the following layer [2]. Many times the connectivity includes paths between non-consecutive layers, and even feedback connections are present.

Vision sensing and object recognition in brain is not processed frame by frame; it is processed in a continuous way, spike by spike, at the brain-cortex. The visual cortex is composed by a set of layers [1-2], starting from the retina. The processing stage starts when the retina captures the information. In recent years, there have been significant progresses at the field of visual cortex processing. Many artificial systems that implement bio-inspired software models use biological-like processing that outperform more conventionally engineered machines [3-4][10]. However, these systems generally run at extremely low speeds because their models are implemented as software programs. For real-time solutions, a direct hardware implementation is required. A growing number of research groups world-wide are implementing some of these computational principles onto real-time spiking hardware through the development and exploitation of the so-called AER (Address Event Representation) technology.

AER was proposed by the Mead lab in 1991 [5][7] for communications between neuromorphic chips using spikes. Every time a cell on a sender device generates a spike, it transmits a digital word representing a code or address for that pixel, using an external inter-chip digital bus (the AER bus). In the receiver the spikes are directed to the pixels whose code or address was on the bus. In this way, cells with the same address in the emitter and receiver chips are virtually connected by streams of spikes. Arbitration circuits ensure that cells do not access the bus simultaneously (see Figure 1). Usually, these AER circuits are built using self-timed asynchronous logic [6].

Several works are already present in the literature regarding the spike-based visual processing filters. Serrano et al. presented a chip-processor able to implement image convolution filters based on spikes that work at very high performance parameters (~3GOPS for 32 × 32 kernel size)

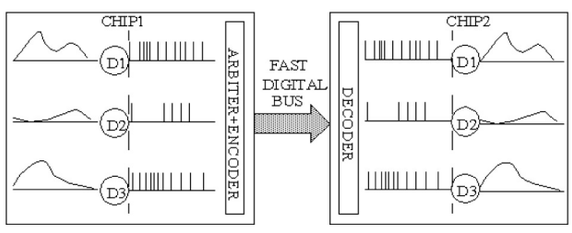


Figure 1 AER Inter-Chip Communication Scheme

*Corresponding author: Manuel Domínguez-Morales; E-mail: mdominguez@atc.us.es

compared to traditional digital frame-based convolution processors [10-12].

There is a community of AER protocol users for bio-inspired applications in vision and audition systems, as demonstrated by the success in the last years of the AER group at the Neuromorphic Engineering Workshop series [22]. One of the goals of this community is to build large multi-chip and multi-layer hierarchically structured systems capable of performing complicated array data processing in real time. The power of these systems can be used in computer based systems under co-processing. This purpose strongly depends on the availability of robust and efficient AER interfaces [9]. One such tool is a PCI-AER interface that allows not only reading an AER stream into a computer memory and displaying it on screen in real-time, but also the opposite: from images available in the computer's memory, generate a synthetic AER stream in a similar manner a dedicated VLSI AER emitter chip [6-7] would do. This PCI-AER interface is able to reach up to 10Mevps (Mega events per second) bandwidth, which allows a frame-rate of 25 frps (frames per second) with an AER traffic load of 100% for 128 × 128 frames, and 25 fps with a typical 10% AER traffic load.

In [19] we evaluated the performance of several frame-to-AER conversion software methods for real-time video applications by measuring execution times in several processors. That work demonstrated that for low AER traffic loads any method in any CPU achieved real-time, but for high-bandwidth AER traffics, it depends on which method and CPU are selected in order to obtain real-time. In this work we focus on the best processor of that study and then we analyze the reasons that made some of the methods non real-time adequate. In this work we analyze the performance of a whole AER co-processor system for visual information processing. It is composed by a PC running a software application that converts frames to AER, a PCI-AER interface that sends the produced AER stream to an AER convolution processor that implements visual filtering on a Xilinx Virtex-5 FPGA.

Next section presents the complete architecture composed by (a) the software methods for converting digital frames into AER format in the computer's memory; (b) the PCI-AER architecture and (c) the AER convolution processor. This section also discusses the performance lacks of the hardware components of the system (b and c). Then in Section 3 we focus on the performance limits of (a) using hardware internal counters of Intel processors. In Section 4 we present the conclusions.

2 System Architecture

Figure 2 shows a block diagram of the architecture under performance evaluation. From left to right there is

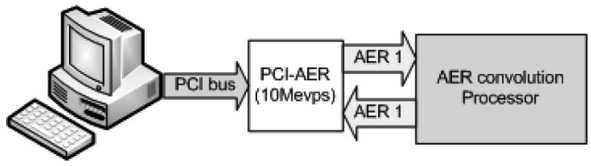


Figure 2 Co-processor Architecture Block Diagram

a computer executing the conversion routines from static images of a video source to a spikes stream into AER format, stored in the main memory of the computer. These streams of AER are moved to the internal FIFO of a PCI board using Direct Memory Access (DMA) with burst PCI bus cycles. Up to 10 Mevps sustained bandwidth has been reported in [22] as output AER rate. The last element of the architecture processes the input AER stream to extract features, like edges, shapes …. A sequence of events in the output of this AER processor represents the result of the programmed convolution for the input visual information. For example, for identifying an object in the visual information the output should be an event that corresponds with the location of the object in the space.

2.1 Frame-to-AER Software Methods

Many software algorithms to transform a bitmap image (stored in a computer's memory) into an AER stream of pixel addresses can be thought [8]. In all of them the address appearance frequency of a concrete pixel must be proportional to its intensity at the image. It is important to denote that the locations of the address pulses are not critical. The pulses can be slightly shifted from their nominal positions; the AER receivers will integrate them to recover the original pixel waveform.

Whatever algorithm is used, it will generate a vector of addresses that will be sent to an AER receiver chip via an AER bus. Let us call this vector the *frame vector*. The *frame vector* has a fixed number of *time slots* to be filled with event addresses: *Tframe/Tpulse*. The number of *time slots* depends on the time assigned to a frame (for example *Tframe* = 40 ms) and the time required to transmit a single event (for example *Tpulse* = 10 ns). If we have an image of $N \times M$ pixels and each pixel can have a grey level value from 0 to K, one possibility is to place each pixel address in the *frame vector* as many times as the value of its intensity, and distribute it at equidistant positions. If all pixels have the maximum value K, the frame vector would be filled with $N \times M \times K$ addresses. This number should be equal to the total number of *time slots* in the *frame vector*. Depending on the total intensity of the image there will be more or less empty slots at the *frame vector*. Each algorithm would implement a particular way of distributing these address events, and will require a certain computational time. This time should be lower than 40 ms in order to warranty real-

time processing. Once the *frame vector* is filled, it is sent to AER convolution processor by the PCI-AER interface. These two hardware components represent a second stage of a pipeline structure; the first one is the frame-to-AER selected method. Therefore, hardware components cannot consume more than 40ms for transferring the *frame vector*.

In [8], several methods for converting video frames into AER were presented. In this work we will focus on a set of them:

- The *Uniform method* distributes equidistantly the events of one pixel along the *frame vector*. The image is scanned pixel by pixel only once. For each pixel, the generated pulses must be distributed equally along the vector. When the *frame vector* is getting filled, the algorithm will find collisions. A collision occurs when an event has to be placed at a *time slot* that is already occupied. In this case, the event is moved to the nearest empty slot of the *frame vector*.
- The *Random method* distributes randomly the events of one pixel along the *frame vector*. The image is also scanned only once. For each pixel, a Linear Feedback Shift Register (LFSR) pseudo-random number generator is used in order to obtain the time slot position at the *frame vector*. The LFSR [18] mathematical properties ensure that no collision will appear for the whole image.
- The *Random-Square method* distributes the events randomly but dividing the frame vector in K slices (one slice per gray level). A double LFSR is used. First one (*LFSR-slice*) is used to select a slice and the second one (*LFSR-pixel*) selects a *time slot* inside the slice. The slice and pixel addresses are joined to place an event at the *frame vector*.
- The *Random Hardware method* also distributes the events randomly at the *frame vector*. In this case the *frame vector* is filled in order, sequentially, with randomly selected events from the bitmap image. A LFSR of $log(N) + log(M) + log(K)$ bits size is used to select an image pixel address and a gray level to decide if an event is placed or not at the *frame vector*. The computational costs of this event are always the same because every *time slot* of the *frame vector* is processed.
- The *Exhaustive method* divide the *frame vector* into K slices and pre-assigns a position for each pixel address at the slice. The events for a certain pixel address are placed at the *frame vector* using equidistant slices. These slices are calculated using modulus operations. The image is scanned only once.
- The *Scan method* goes over the image row by row as many times as the maximum gray level. The frame vector is filled in order from the beginning. For each pixel, *Scan method* compares the gray level to zero. When the pixel value is greater than zero, an AER event is placed at the next *time slot* and the gray level of the present pixel is decremented by one. If the gray level is zero the next *time slot* is left empty.

2.2 Hardware PCI-AER Bridge

This hardware interface has to be able to send up to NxMxK events in 40 ms. Depending on the interface performance, $N \times M$ (image sizes) and K value (gray levels), this could be real or not.

The PCI bus [21] has several bandwidth versions depending on the bus clock frequency and the data bus width. Clock speed could be 33 MHz or 66 MHz, and the data bus length could be 32 bits or 64 bits. There is an improved version of the parallel PCI called PCI-X that can work up to 133 MHz. The shown interface uses the most common PCI bus, at a speed of 33MHz and 32 bits of bus size.

When we use 32 bits to represent one AER spike, the maximum theoretical peak rate of 33Mevps could be achieved. Nevertheless, since a PCI bus cycle spends some time for arbitration, and usually there are other PCI boards sharing the PCI bus in a standard computer, a real event rate over the PCI bus of 10 Mevps has been obtained and reported on [22].

Therefore, a minimum inter-spike-interval (ISI) of 100 ns is present on the AER output of the PCI-AER bridge. So, in 40 ms, the AER bus can transmit a 400 K events rate maximum. If the methods described in the previous sub-section are working with 128 × 128 images of 256 gray levels and these frame-to-AER methods uses as many events as the gray level, a maximum amount of 128 × 128 × 255 ≈ 4,17Mevents can be produced for each frame (104,45 Mevps is required). So, the PCI-AER interface reduces the real-time capabilities to those bitmap images or video information represented in AER that implies a 10% of AER traffic.

PCI-AER bridge consists (see Figure 3) into a specialized CORE that interacts with the PCI bus. A set of configuration registers are necessary for PCI protocol, like BAR (based address registers), …. Two independent First-Input-First-Output (FIFO) memories works in parallel in

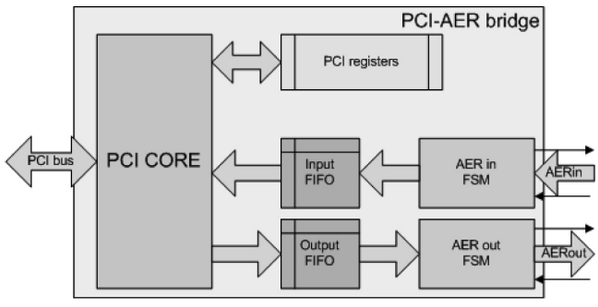


Figure 3 PCI-AER Bridge Architecture Block Diagram

order to allow the AER traffic to go in both ways (in and out) of the computer. AER-in-FSM component manages the input AER protocol and stores received events values and a time-stamp in the input-FIFO. Once the input-FIFO has enough events for initiating a PCI burst bus cycle, the PCI CORE sends to the computer's main memory the received sequence of events.

In contrast, when there is a sequence of events in the computer's memory ready to be sent to the AER system, a PCI burst transaction is initiated for transferring that event sequence to the output-FIFO of the PCI-AER bridge. Each stored event at the output FIFO has information about the event address to be transmitted and the wait time since last transmission (time-stamp).

This interface is completely based on a Spartan II 200 FPGA using the 66 MHz clock inside the FPGA, obtained by multiplying by 2 the PCI clock.

2.3 AER Convolution Processor

This block processes AER information coming from a video source in order to extract features of the visual information like edges, noise reduction, objects detection, or any other feature that could be implemented using convolutions.

The main advantage of spike-based processing is that the result of the operation is coming out only a few nanoseconds after the first incoming events. In contrast to digital frame based image filters, AER processors start calculating the output when the first event arrives, and they do not have to wait for the whole image or frame before start processing.

A convolution operation applied to a bitmap image is defined mathematically as follows:

$$\forall_{i,j} \rightarrow Y(i,j) = \sum_{a=-n/2}^{n/2} \sum_{b=-m/2}^{m/2} K(a,b) \cdot X(a+i, b+j)$$

Where X is the input image; Y is the output image; K is the kernel; (i, j) represents a pixel position and (n, m) is the image dimension.

If the information is codified in spikes, the convolution can be processed by implementing each multiplication as a sequence of additions [10]. Every time an AER input is received, an amount is added to a neighborhood of the resulting image. This amount depends directly on the weights of a pre-programmed kernel. Next equation represents this behavior for a given input event (i, j):

$$Y(i+a, j+b) = Y(i+a, j+b) + K(a,b), \ \forall a,b \in \dim(K)$$

This means that, for an input event, the convolution kernel is added to the neighbor's pixels of the input address. When all the events have been received for a particular pixel, the resulting pixel is the accumulation of multiplication of the gray level of the neighbors and the kernel value.

Independently of that, if one pixel of the resulting image reaches a threshold, an output spike is generated and the pixel is reset. In this way those pixels that produce output events correspond to the result of the convolution performed in this processor.

Figure 4 presents the block diagram of the AER convolution processor. A Finite State Machine manages the input traffic, accesses the results memory and the kernel one, and generates output events in an output FIFO. A forgetting mechanism is implemented to reset those pixels with sporadic traffic.

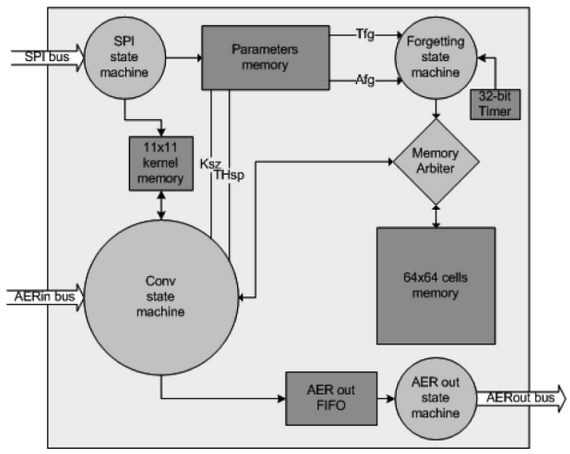
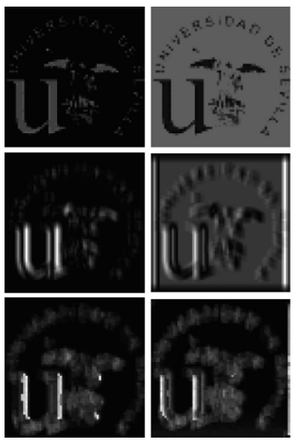


Figure 4 FPGA AER Convolution Processor Block Diagram

Figure 5 shows an example. In the first row there are two different input images. The convolution kernel for extracting vertical edges is K. Middle row shows the convolution results using the MATLAB conv2() function. Last row shows the histograms of the output events obtained from the AER convolution processor with the kernel K. The input is an AER stream obtained by translating the first row images using the uniform method explained in Sub-section 2.1.

This convolution processor performance [10] depends directly on the kernel size of the convolution to be performed. For 3 × 3 convolution kernels, an input event rate of 3.85 Mevps is processed without introducing delays. But, for the maximum supported kernel size of 11 × 11, this processor can only support an input event rate of 1.35 Mevps.

For each input event, the kernel is processed row by row; so the bigger is the kernel, the worse is the performance of the processor. This can be improved applying parallel techniques for processing the whole kernel in parallel, or improving the technology in order to



$$K = \begin{pmatrix} -10 & 0 & 10 \\ -10 & 0 & 10 \\ -10 & 0 & 10 \end{pmatrix}$$

Figure 5 AER Convolution Processor Example

increase the clock frequency of the system from 50MHz to other faster.

3 Frame-to-AER Performance Study

Presented co-processor system has several lacks in order to be competitive for real-time applications. Hardware components (PCI-AER bridge and AER convolution processors) need architectural and technological improvements for real-time capabilities.

On the other hand, the frame-to-AER software methods performance is the most critical issue, because, as presented in [19][23], using an Intel Core 2 Quad processor, several of the methods cannot reach a sustained rate of 25 frames conversions per second.

In this section we present a performance study of these methods analyzing the internal performance counters available on Intel processors for measuring several parameters for identifying the lacks of each method. Internal performance counters can be pre-programmed to measure a particular parameter before the execution of a process or software. These parameters can be branch miss predictions, cache miss (both L1 and L2), cache store line blocks, etc.

The Intel Core 2 Quad processor consists of four blocks composed by a core processor and a L1 separate cache of 2 × 32 KB, 8 ways and 64 bytes line and two L2 caches of 4 MB, 16 ways and 64bytes line. Each one is shared by the two cores and L1s. This multi-core architecture presents coherence conflicts between L1 lines due to the possibility of having parallel processes working with the same L1 data line in different cores. This requires hard penalties for updating shared lines between L1 caches using L2 as interchange mechanism.

If the frame-to-AER methods can be divided in parallel threads working with independent data, then synchronizations between L1 lines would not be needed and therefore, these cache penalties would not appear, improving execution times.

In order to analyze the penalties of these L1 and L2 problems, and other possible problems, during the execution of the frame-to-AER methods we propose to use VTune. VTune [17] is an Intel application designed to measure the performance of a set of parameters of Intel processors by using internal hardware counters. There are two internal counters for measuring a set of parameters, so several executions are needed. These counters must be configured previously for analyzing two different parameters for each execution. VTune allows preparing a project for performance analysis in which several executions of the application under study are launched automatically.

We have selected the following parameters [16]:

- **L2_LINES_IN.SELF.ANY (L2 cache misses):** This event counts the number of cache lines allocated in the L2 cache. Cache lines are allocated in the L2 cache as a result of requests from the L1 data and instruction caches and the L2 hardware prefetchers to cache lines that are missing in the L2 cache. This event has been configured to count occurrences for all cores.

Those methods that cannot be divided into parallel threads that are working in different parts of the array used to store the generated events will increase this parameter.

- **STORE_BLOCK.ORDER (L1 data cache and DTLB stall events):** Intel processors maintain an in-order writing of results in cache. Therefore, although instructions are executed following an out-of-order architecture (dynamic scheduling), results are re-ordered before their writing operations to cache or registers. This mechanism is supported by the Re-Order-Buffer that also solves miss-predictions in branches or interruptions. When results are written in cache by one core, this core writes without any synchronization respect to other cores. Therefore it is possible that a store executed in one core implies to write a line into L2 cache and this line is dirty because it is being used by another core L1 cache. *This event counts the total duration, in number of cycles, which stores are waiting for a preceding stored cache line to be observed by other cores*. In general, when increasing the number of threads working with the same part of the memory, this parameter should increase.
- **L1D_CACHE_LOCK.MESI (L1 data cacheable locked reads):** This event counts the number of locked data reads from cacheable memory. In the Intel Core 2 Quad, two cores share L2 cache. If these two cores are working with the same L2 cache line, each one of them has a copy on their own L1 caches. Therefore, if a core modifies its L1 line, the other core corresponding L1 line must be invalidated. This is the functionality of the MESI protocol [15]. When this occurs and the second core

needs to access its shared L1 line, it needs to reload the L1 line from the L2, but it is probable that L2 line has not been updated by first core L1, so the penalty is increased. These situations appear more frequently when threads of the same process are sharing memory, increasing the synchronization between threads, which occurs to the AER methods.

- **L1D_CACHE_LOCK_DURATION (Duration of L1 data cacheable locked operation):** This event counts the number of cycles during which any cache line is locked by any locking instruction. Locking happens at retirement and therefore the event does not occur for instructions that are speculatively executed. Locking duration is shorter than locked instruction execution duration.
- **RESOURCE_STALLS.BR_MISS_CLEAR (Cycles stalled due to branch miss-prediction):** This event counts the number of cycles after a branch miss-prediction is detected at execution until the branch and all older micro-ops retire. During this time new micro-ops cannot enter the out-of-order pipeline.

In order to analyze the performance an executable file for each AER method and for each input image has been prepared. The test images set (TIS) selected is shown in Figure 6. All the images have been constructed randomly, with a Gaussian histogram and for producing different bandwidth of events in the AER bus (from 10% to 90% and 95, 97, 99%)

Figure 7 shows graphically the results obtained when monitoring the events presented using the internal hardware counters of the Intel Core 2 Quad processor.

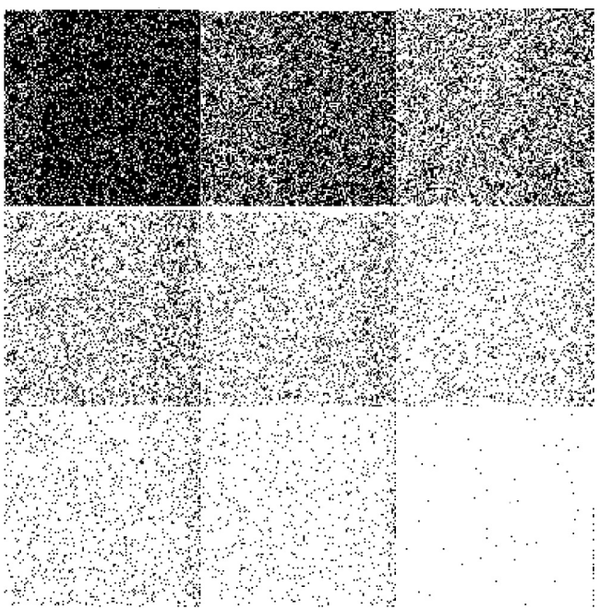


Figure 6 TIS Generated Randomly to Have Gaussian Histogram. Resulting Images (10% Load Upper Left, 90% Load Lower Right)

For several methods, the more events are produced in the AER bus, the greater the number of L2 lines required (see L2_LINES_IN.SELF.ANY graph). These methods are Random, Random OMP, Uniform OMP and Exhaustive OMP. OMP means that OpenMP compilation techniques have been used. These OMP methods require around 24 threads that require an increment of synchronization points between threads when producing AER events. These synchronization points increment imply a data replication between both the two L2 caches and the four L1 caches, because several threads are accessing same parts of the frame period.

The number of misses produced when accessing L2 for storing results is higher for Random and Random OMP methods. This means that for these methods, no matter the number of threads, collisions in cache are produced for different cores more frequently than for the other methods. This effect is due to the fact of sharing not only the frame vector, but also the rand function used in software. This function must share a global variable because in other case it cannot be ensured the property of LFSR in producing all the positions randomly without repetition.

L1D_CACHE_LOCK.MESI measures the number of L1 lines invalidated by another core when lines are shared between several cores. This situation is also more accentuated in Random methods due to rand function sharing.

Duration of L1D locks does not show significant differences between methods.

Branches miss-prediction occurs more frequently for Random methods due to the difficulty in predicting the behavior of a random sequence. But this miss-predictions are also higher for OMP methods compared to not OMP ones. This is normal taking into account the difference in the number of threads, each of them requiring different BTB entries (Branch Target Buffer).

We have seen that the software methods using OMP techniques are not adequate for Intel Core 2 Quad, and that Random methods could be not feasible for real-time visual processing. They did not reach real-time capabilities because of the rand function sharing.

The solution is to divide the image in as many parts as cores has the processor (N cores), and use N different and adapted rand functions for generating the sequence of events for each part of the image and then join all of them in the *frame vector*.

A new method called Random Quadrant divides the LFSR in four parts for the Intel Core 2 Quad.

Figure 8 shows execution times in the same Intel Core 2 Quad processor of the Random Quadrant method, compiled using Intel compiler, Random and Random Square without OMP techniques.

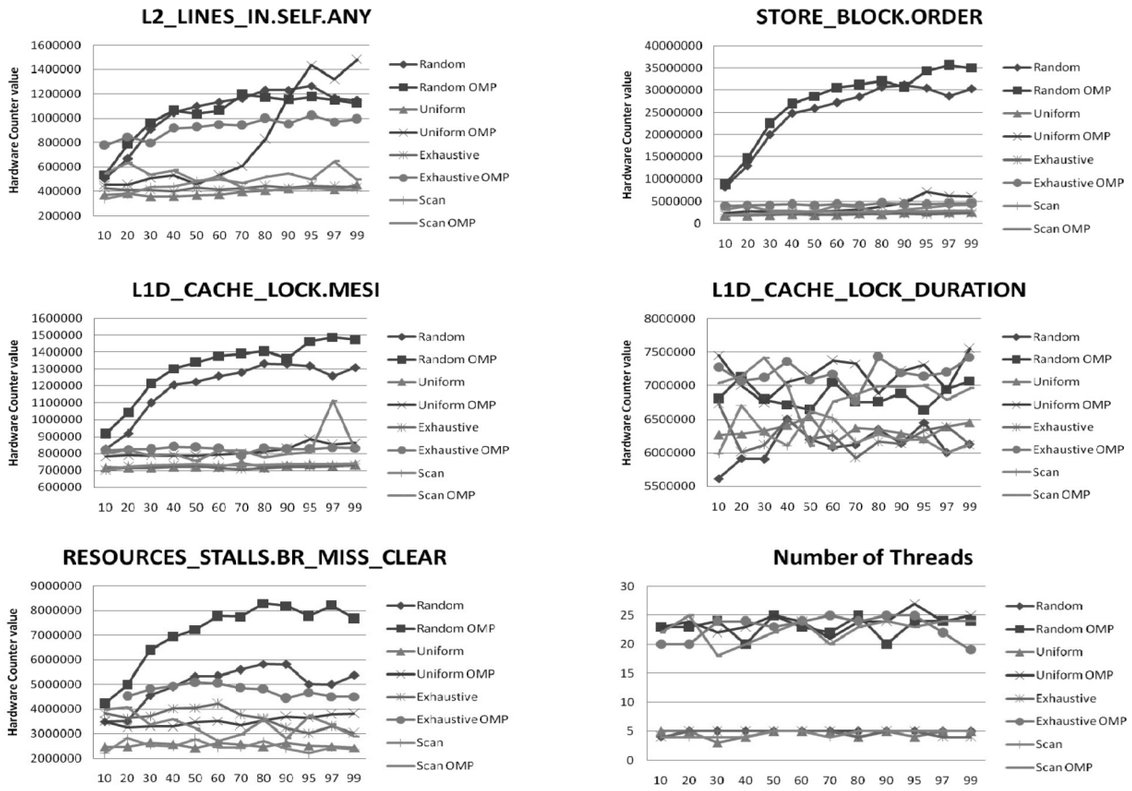


Figure 7 Intel Core 2 Quad Internal Events Performance Values for Executions of Frame-to-AER Methods Using TIS Images (10-99% AER Bus Occupation)

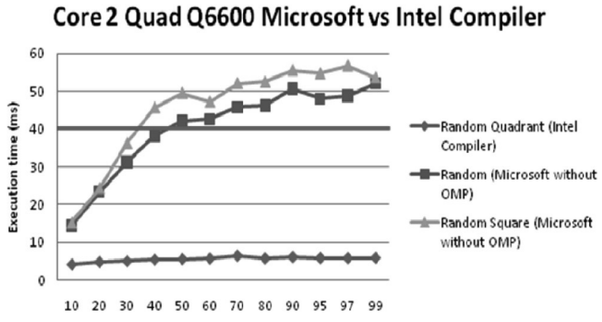


Figure 8 Intel Core 2 Quad Execution Times for Random, Random Square and Random Quadrant Method for TIS Images

4 Conclusions

We can conclude with this performance study that hardware PCI-AER bridge and AER convolution processor needs architectural and/or technological improvements to achieve real-time (25 fps).

On the other hand, regarding the software, we have analyzed several internal CPU counters. We have demonstrated that random methods are not able to achieve real-time in multi-core processors.

Finally, we have proceeded to modify the random method to solve these inefficiencies, achieving a substantial performance improvement.

References

[1] Daniel Drubach, *The Brain Explained*, Prentice Hall, Englewood Cliffs, NJ, 2000.

[2] Gordon M. Shepherd, *The Synaptic Organization of the Brain* (3rd ed.), Oxford University Press, New York, 1990.

[3] J. S. Lee, *A Simple Speckle Smoothing Algorithm for Synthetic Aperture Radar Images*, *IEEE Trans. on Systems, Man and Cybernetics*, Vol.13, No.1, 1983, pp.85-89.

[4] Thomas R. Crimmins, *Geometric Filter for Speckle Reduction*, *Applied Optics*, Vol.24, No.10, 1985, pp.1438-1443.

[5] Massimo Antonio Sivilotti, *Wiring Considerations in Analog VLSI Systems with Application to Field-Programmable Networks*, Ph.D. Thesis, California Institute of Technology, Pasadena, CA, 1991.

[6] Kwabena A. Boahen, *Communicating Neuronal Ensembles between Neuromorphic Chips*, in T. S. Lande (Ed.), *Neuromorphic Systems Engineering*, Kluwer Academic, Boston, 1998, pp.229-259.

[7] Misha Mahowald, *VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function*, Ph.D. Thesis, California Institute of Technology, Pasadena, CA, 1992.

[8] A. Linares-Barranco, G. Jimenez-Moreno, A. Civit-Ballcels and B. Linares-Barranco, *On Algorithmic Rate-Coded AER Generation*, *IEEE Transactions on Neural Networks*, Vol.17, No.3, 2006, pp.771-788.

[9] R. Paz, F. Gómez-Rodriguez, M. A. Rodriduez, A. Linares-Barranco, G. Jimenez and A. Civit, *Test Infrastructure for Address-Event-Representation Communications*, *Proc. IWANN 2005*, Barcelona, Spain, June, 2005, pp.518-526.

[10] A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodriguez, A. Jiménes, M. Rivas, G. Jiménes and A. Civit, *On the AER Convolution Processors for FPGA*, *Proc. ISCAS 2010*, Paris, May, 2010, pp.4237-4240.

[11] Ben Cope, *Implementation of 2D Convolution on FPGA, GPU and CPU*, http://cas.ee.ic.ac.uk/people/btc00/index_files/Convolution_filter.pdf

[12] Ben Cope, P. Y. K. Cheung, W. Luk and S. Witt, *Have GPUs made FPGAs redundant in the field of video processing?*, *Proc. FPT 2005*, Singapore, Singapore, December, 2005, pp.111-118.

[13] C. Farabet et al., *CNP: An FPGA-based Processor for Convolutional Networks*, *FPL 2009*, Prague, Czech Republic, August, 2009, pp.32-37.

[14] N. Farriga et al., *Design of a Real-Time Face Detection Parallel Architecture Using High-Level Synthesis*, *Hindawi Publishing Corporation. EURASIP Journal on Embedded Systems*, Vol.2008, doi:10.1155/2008/938256.

[15] *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, 2008, http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html

[16] *Intel Architecture Software Developer's Manual. Volume 3: System Programming*, 1999, http://www.scribd.com/doc/5510805/Intel-Architecture-Software-Developers-Manuals-Volume-3-System-Programming-

[17] James Reinders, *VTune™ Performance Analyzer Essentials: Measurement and Tuning Techniques for Software Developers*, Intel Press, Santa Clara, CA, 2005.

[18] Xilinx Inc., *Linear Feedback Shift Register V2.0*, 2001, http://www.xilinx.com/ipcenter

[19] Manuel Domínguez-Morales, Pablo Iñigo-Blasco, et al., *Performance study of synthetic AER generation on CPUs for Real-Time Video based on Spikes*, *Proc. SPECTS 2009*, Istambul, Turkey, July, 2009, pp.57-64.

[20] Barbara Chapman et al., *Using OpenMP. Portable Shared Memory Parallel Programming*, The MIT Press, 2007.

[21] *Conventional PCI 3.0 -- An Evolution of the Conventional PCI Local Bus Specification*, 2012, http://www.pcisig.com/specifications/conventional/pci_30/

[22] Avis Cohen et al., *Report on the 2006 Workshop on Neuromorphic Engineering*, 2006, http://www.ine-news.org/view.php?article=pdf_20071107060101

[23] Manuel Domínguez-Morales et al., *Frames-to-AER Efficiency Study Based on CPUs Performance Counters*, *Proc. SPECTS 2010*, Otawa, Canada, July, 2010, pp.141-148.