Performance Study of Software AER-Based convolutions on a Parallel Supercomputer

Rafael J. Montero-Gonzalez[1], Arturo Morgado-Estevez[1], Alejandro Linares-Barranco[2], Bernabe Linares-Barranco[3], Fernando Perez-Peña[1], Jose Antonio Perez-Carrasco[3], and Angel Jimenez-Fernandez[2]

[1] Applied Robotics Research Lab, Engineering School, University of Cadiz, Spain
C/Chile 1, 11002-Cádiz
{rafaeljesus.montero,arturo.morgado,fernandoperez.pena}@uca.es [2]

Robotic and Technology of Computers Lab, University of Seville, Spain
Av. Reina Mercedes s/n, 41012-Seville
{alinares,ajimenez}@atc.us.es

[3] Institute of Microelectronics of Seville, IMSE-CNM-CSIC, Spain
Calle de los Descubrimientos, Pabellón Pza. de América, 41092-Seville
{jcarrasco,bernabe}@imse-cnm.csic.es

Abstract. This paper is based on the simulation of a convolution model for bio-inspired neuromorphic systems using the Address-Event-Representation (AER) philosophy and implemented in the supercomputer CRS of the University of Cadiz (UCA). In this work we improve the runtime of the simulation, by dividing an image into smaller parts before AER convolution and running each operation in a node of the cluster. This research involves a test cases design in which the optimal parameters are set to run the AER convolution in parallel processors. These cases consist on running the convolution taking an image divided in different number of parts, applying to each part a Sobel filter for edge detection, and based on the AER-TOOL simulator. Execution times are compared for all cases and the optimal configuration of the system is discussed. In general, CRS obtain better performances when the image is divided than for the whole image.

Keywords: AER, convolution, parallel processing, cluster, supercomputer, bio-inspired, AER simulator.

1 Introduction

Nowadays computer systems are increasing their performance looking for the solution of real-world problems using models inspired in biology. These systems, called bio-inspired systems, analyze the operation of parts of the body and try to implement it in a similar manner through electronic and/or computer systems.

Address-Event-Representation systems are composed of sets of cells typically distributed in a matrix that process the information spike by spike in a continuous

[*] This work has been supported by Spanish Government grant VULCANO (TEC2009-10639-C04-02) of the Minister of Science and Innovation.

way. The information or results of each cell is sent in a time multiplexed strategy using a digital bus, indicating which position is producing the event.

If we represent a black and white image as an array of cells where each pixel value is in gray scale, the white level would correspond to a frequency value determined by allocating the largest amplitude values, higher brightness values. The signal caused by each pixel is transformed into a train of pulses using PFM (pulse frequency modulation) [1].

Based on the interconnection of neurons present on human vision, the continuous state of transmission in a chip is transformed into a sequence of digital pulses (spikes) of a minimum size (of the order of ns) but with an interval between spikes of the order of hundreds of microseconds (us) or even milliseconds (ms). This interval allows time multiplexing of all the pulses generated by neurons into a common digital bus. Each neuron is identified with an address related to its position into the array. Every time a neuron emits a pulse, its address will appear in the output bus, along with a request signal, until acknowledge is received (handshake protocol). The receiver chip reads and decodes the direction of incoming events and issues pulses for the receiving neurons.

One of the operations performed by AER systems, applied to artificial vision, is the convolution. The first operations in the brain cortex consist of convolution for object edges detection, based on calculations of brightness gradients. In the design presented in [2], a system is described where a single convolution processor performs all operations for the whole image.

Based on this idea, and the divide and conquer premise, this paper is arguing that the division of the image into smaller parts before AER convolution processing in parallel will reduce the runtime. With this new design a convolution could be proposed where a multiprocessor system may perform operations in less time.

2 Methodology and Test Cases

The process of experimentation is to verify, through an exhaustive analysis, which would be the different runtimes of the convolution of an image. Each runtime will correspond to different divisions. All division convolutions are performed in parallel, instead of performing the convolution of the whole image.

We have used the Cluster of Research Support (CRS), part of the infrastructure of the UCA, for improving execution times of the simulation tool AER TOOL. In order to run this simulator on CRS we propose a new simulation model parameterized and adapted to running tests in parallel processors.

2.1 Supercomputer CRS (Cluster of Research Support)

The CRS is composed of 80 nodes. Each node has 2 Intel Xeon 5160 processors at 3 GHz with 1.33GHz Front Side Bus. Each processor is Dual Core, so we have 320 cores available. A total of 640GB of RAM memory, 2.4TB of scratch and Gigabit Ethernet communication architecture with HP Procurve switches allow to obtain a peak performance of 3.75 TFLOPS [3].

In terms of software features, to manage distributed work, Condor[1] tool is used. Condor is a job manager system that specializes in calculation-intensive tasks. The coding for the simulation was done using MATLAB and AER TOOL simulator [4] for MATLAB.

Developing this set of tests on a real physical implementation would be highly expensive. The supercomputer CRS provides the possibility of an AER simulation model implementation in parallel with acceptable runtimes, using the software installed and existing libraries.

2.2 Test Image and Successive Divisions

For this simulation we have designed an image in Adobe Photoshop CS, using gray scale, where the pixel having the darkest value will have a value close to 0 and the brightest will be close to 255. The GIF image size is 128x128 pixels of 0-255 gray levels.

The idea of dividing the original image and perform parallel convolution arises from trying to take advantage of distributed processing systems to expedite the process. This involves running a series of tests with different numbers of divisions.

Firstly, we have obtained the process runtimes of the convolution of the original image without divisions. Secondly the image has been divided into 4 parts (64x64 bits each), performing the convolution in a different processor. Then, the sequence has been repeated by 16 divisions (32x32 bits each). Next, using 256 divisions (8x8 bits each), and finally we have concluded with 1024 divisions (4x4 bits each). Conceptually, the operation would be as shown in Fig. 1.

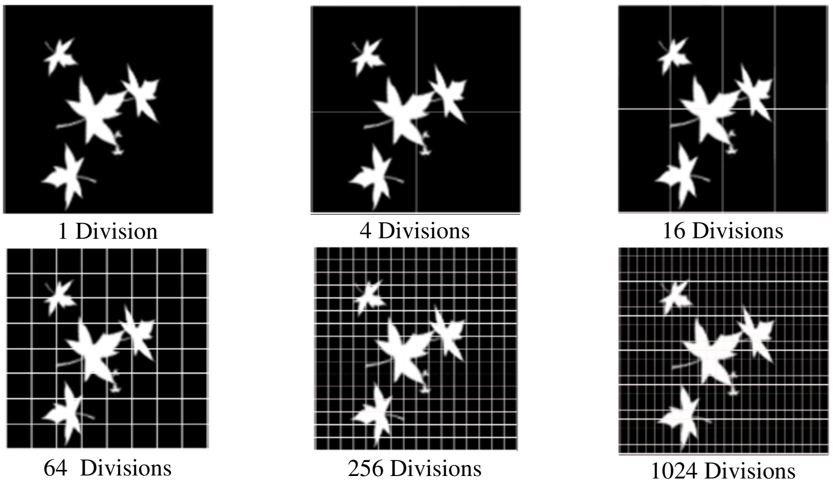


Fig. 1. Divisions of the original image to simulate

[1] http://www.cs.wisc.edu/condor/

Edge detection operation by convolution was performed at each division in a different node, estimating that for smaller image size the runtime will be reduced.

2.3 Topology Diagram Implementation

For this research, parametric model simulation software has been developed, whose test cases are specified by variable assignment.

Once the simulation variables are set, the system runs following the block diagram shown in Fig. 2.

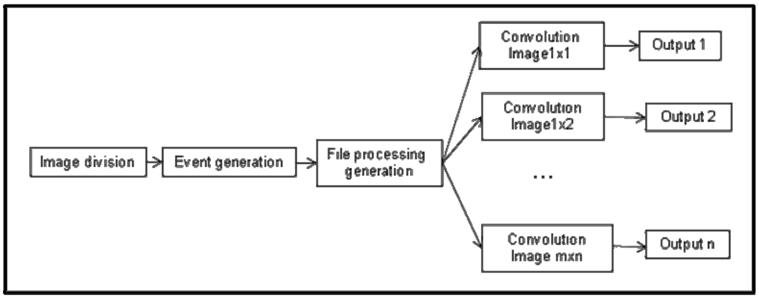


Fig. 2. Simulation block diagram

First, the division of the image is performed using specified parameters. Then, the Uniform method [2] was used for events generation algorithm. When applying this algorithm, a minimum time interval between consecutive events of 0.2 ms and a maximum of 400K events per frame are specified. The next step generates all files necessary for processing the AER TOOL in the CSR cluster. Then, the convolution filter is performed for each division on a different node. Finally, we got as many outputs as image divisions were generated, with the result of applying the operation.

For the convolution filter Sobel edge detection was used in horizontal averaging the diagonal values of a 3x3 size.

$$S = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}. \tag{1}$$

Parameters that have been considered for the study are:

- Number of cores: 4, 8, 16 and 32.
- Number of divisions of the image:
  - 1 image of 128x128 pixels.
  - 4 divisions of 64x64 pixels.
  - 16 divisions of 32x32 pixels.
  - 64 divisions of 16x16 pixels.
  - 256 divisions of 8x8 pixels.
  - 1024 divisions of 4x4 pixels.
- Convolution matrix: Sobel of 3x3.

Once we have recorded the runtimes of each stage, analyzed the graph generated and detected the highest peak on the surface, we can indicate the optimal design for the system.

3 Results and Discussion

CSR cluster is a shared computational resource at UCA. Execution times may depend on the cluster workload and users. A variation in the order of milliseconds has been detected. In order to minimize these undesirable situations we have selected a low workload day (Saturday) and a reduced number of nodes respect to the maximum available number of nodes in the cluster. Tests were performed 3 times and the averaged execution times are represented in tables 1- 4 and their respective figures.

The test took place on 9/10/2010 with a workload of 30% consumed by other 9 users running their own independent application of this test.

Processing time for each stage and the total can be seen in the following tables, expressing all the time in seconds for each number of nodes. Data movement time haven't considered because it's not possible to access using user privileges in cluster management.

Table 1 presents both the event generation and the convolution execution times for selected image divisions and using 4 nodes (16 cores) of the CSR. It can be observed that there is no significant difference for 1 or 4 divisions. Nevertheless, for 64 or 256 divisions, runtimes are doubled and a significant difference for 1024 divisions can be seen. However, when generating events it can be seen that the lower is the number of divisions, the higher is the execution time, except for 1024 divisions. In the case of parallel execution it can be seen that leaving the image on its original size and dividing it into 4 pieces of 64x64 has a significant time difference too. It can be also observed that there is a runtime increment for 64 image divisions. For the total runtime (table 5, 4 nodes column), the best execution times correspond to 64 divisions.

Table 2 presents corresponding runtime results when tasks are scheduled for 8 nodes of the cluster. Now it can be seen that runtimes are improved in general terms, but these results do not imply significant changes. For the image division task, the lowest execution time remains for 1 division. For the event generation task, the lowest result is obtained for 256 divisions. And for the convolution task, runtime is also the lowest for 4 divisions, like for 4 nodes.

In Table 3 runtime results correspond to the use of 16 nodes of the cluster. Image division task has similar results than for lower number of nodes. Event generation task runtime offers significant changes for 8x8 blocks (when divided into a total of 256 images), but their convolution runtimes do not produce improvements. In the parallel execution of convolutions, it is found that 64x64 divisions have reduced runtime. For 32x32 and 16x16 images runtime is very similar, but when you have 8x8 images runtime increases. This increment is due to the coordination of a large number of processors in the cluster that requires more data traffic between them, resulting in an overall implementation delay.

In Table 4 results are presented when 32 nodes of the cluster are used. Image division task runtime and event generation runtime show similar results to those presented for 16 nodes. Parallel convolution task runtimes are improved for 4, 8 and 16 divisions. Therefore increasing the number of nodes working in parallel does not imply runtimes reduction, but for larger number of divisions, runtimes also increase, starting at dawn when they are 64 divisions of blocks of 16x16 pixels and shooting when divisions reach the 1024 block of 4x4 pixels.

Table 1. Runtimes summary for 4 nodes

N. div	Tdiv img	Tgener. Even.	Runtime. paral
1	24 ms	413,9 s	5,6 s
4	26 ms	101,1 s	5,3 s
16	35 ms	59,1 s	7,3 s
64	40 ms	38,8 s	13,1 s
256	42 ms	20,7 s	37,6 s
1024	94 ms	25,7 s	135,5 s

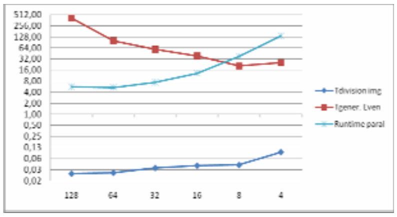


Table 2. Runtimes summary for 8 nodes

N. div	Tdiv img	Tgener. Even.	Runtime paral
1	25 ms	395,6 s	9,3 s
4	30 ms	109,8 s	6,6 s
16	31 ms	62,8 s	7,6 s
64	30 ms	38,8 s	17,0 s
256	43 ms	20,9 s	27,2 s
1024	95 ms	25,8 s	81,5 s

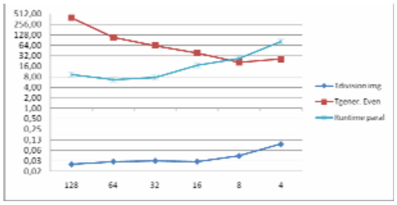


Table 3. Runtimes summary for 16 nodes

N. div	Tdiv img	Tgener. Even.	Runtime paral
1	26 ms	1199,2 s	8,3 s
4	26 ms	171,9 s	5,9 s
16	26 ms	108,9 s	17,4 s
64	28 ms	41,9 s	19,3 s
256	43 ms	22,7 s	34,8 s
1024	96 ms	31,3 s	76,8 s

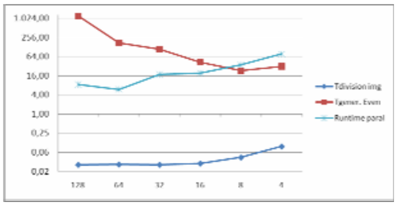
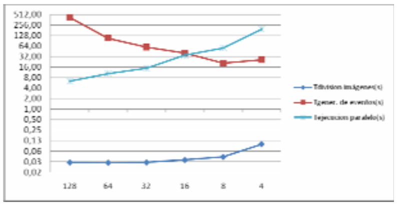


Table 4. Runtimes summary for 32 nodes

N. div	Tdiv img	Tgener. Even.	Runtime paral
1	30 ms	423,7 s	6,3 s
4	30 ms	107,7 s	10,5 s
16	30 ms	59,9 s	14,9 s
64	35 ms	41,0 s	34,8 s
256	43 ms	20,8 s	57,0 s
1024	100 ms	25,8 s	195,6 s



If we represent the total runtime with respect to the maximum number of nodes and the number of divisions, we get Table 5, noting the lowest total runtime shaded.

If instead of using the total runtime, we take the parallel runtime and we represent it in the same domain as Table 5, we obtain Table 6, noting the minimum runtime shaded.

It can be highlighted the case of 4 nodes and 4 divisions of 64x64 pixels blocks which have a faster execution, but not much different block sizes with the 32 or 128.

Table 5. Summary of total runtime as the number of divisions and the number of nodes

N div	4 nodes	8 nodes	16 nodes	32 nodes
1	420 s	405 s	1213 s	430 s
4	107 s	117 s	178 s	118 s
16	67 s	71 s	127 s	75 s
64	52 s	56 s	62 s	76 s
256	59 s	49 s	59 s	79 s
1024	165 s	111 s	113 s	225 s

Table 6. Summary of parallel runtime depending on the number of divisions and the number of nodes

N div	4 nodes	8 nodes	16 nodes	32 nodes
1	5,6 s	9,3 s	8,3 s	6,3 s
4	5,3 s	6,6 s	5,9 s	10,5 s
16	7,3 s	7,6 s	17,4 s	14,9 s
64	13,1 s	17,0 s	19,3 s	34,8 s
256	37,6 s	27,2 s	34,8 s	57,0 s
1024	135,5 s	81,5 s	76,8 s	195,6 s

4 Conclusions

In this work we have designed a test case set for AER convolution processing on a supercomputer, the CSR cluster of UCA, Cadiz, SPAIN. We have executed and compared all the test cases. If we rely on the data obtained we obtain the following conclusions:

- Referring to the data in Table 5, we can see that the total runtime minor by running a maximum of 8 nodes in parallel and 256 divisions of the image into blocks of 8x8 bits. This case is very similar to the case of a maximum of 8 nodes in parallel and 64 divisions of the image into blocks of 16x16 bits. Then, the two implementations would be valid for our system.
- If we look at the data in Table 6, in which only the parallel runtimes are shown, we see that the test case for a maximum of 4 nodes with 4 divisions of 64x64 bits of the image, obtained lower runtimes.
- If we consider that, in a hardware implementation the event generation time disappears when taking images directly from an acquisition event-based system (i.e. silicon retina), the best option is to have 4 nodes in parallel with 4 divisions of 64x64 bits.

This work represents the first steps on the execution of more complex AER system simulations on the cluster, which will improve considerably the performance of parameters adjustment of hierarchical AER systems where several convolution kernels work together in a multilayer system for more complex tasks as face recognition, etc, already illustrated in [7].

Acknowledgments. Andrés Yañez Escolano, member of Languages and Computing Systems Department at University of Cádiz, Spain, and Abelardo Belaustegui González, member of Computing System Area at University of Cádiz, Spain and main manager CRS.

References

1. Serrano-Gotarredona, T., Linares-Barranco, A.G., Andreou, B.: AER image filtering architecture for vision-processing systems. Circuits and Systems I. IEEE Transactions on Fundamental Theory and Applications 46, 1064–1071 (1999)
2. Linares-Barranco, A., Jimenez-Moreno, G., Linares-Barranco, B., Civit-Balcells, A.: On algorithmic rate-coded AER generation. IEEE Transactions on Neural Networks 17, 771–788 (2006)
3. Technical support in supercomputing, University of Cadiz, `http://supercomputacion.uca.es/`
4. Pérez-Carrasco, J.-A., Serrano-Gotarredona, C., Acha-Piñero, B., Serrano-Gotarredona, T., Linares-Barranco, B.: Advanced vision processing systems: Spike-based simulation and processing. In: Blanc-Talon, J., Philips, W., Popescu, D., Scheunders, P. (eds.) ACIVS 2009. LNCS, vol. 5807, pp. 640–651. Springer, Heidelberg (2009)
5. Lujan-Martinez, C., Linares-Barranco, A., Rivas-Perez, M., Jimenez-Fernandez, A., Jimenez-Moreno, G., Civit-Balcells, A.: Spike processing on an embedded multi-task computer: Image reconstruction. In: Fifth Workshop on Intelligent Solutions in Embedded Systems 2007, pp. 15–26 (2007)
6. Camunas-Mesa, L., Acosta-Jimenez, A., Serrano-Gotarredona, T., Linares-Barranco, B.: Fully digital AER convolution chip for vision processing. In: IEEE International Symposium on Circuits and Systems, ISCAS 2008, pp. 652–655 (2008)
7. Perez-Carrasco, J.A., Acha, B., Serrano, C., Camunas-Mesa, L., Serrano-Gotarredona, T., Linares-Barranco, B.: Fast Vision Through Frameless Event-Based Sensing and Convolutional Processing: Application to Texture Recognition. IEEE Transactions on Neural Networks 21, 609–620 (2010)
8. Dominguez-Castro, R., Espejo, S., Rodriguez-Vazquez, A., Carmona, R.: A one-transistor-synapse strategy for electrically-programmable massively-parallel analog array processors. In: 2nd IEEE-CAS Region 8 Workshop on Analog and Mixed IC Design, pp. 12–13, 117–122 (1997)