High-speed analog simulation of CMOS vision chips using explicit integration techniques on many-core processors

Gines Domenech-Asensi
Dpt. de Electrónica, Téc. de Computadoras y Proyectos
Universidad Politécnica de Cartagena
Cartagena, Spain
gines.domenech@upct.es

Tom J. Kazmierski
University of Southampton
Southampton, United Kingdom
tjk@ecs.soton.ac.uk

Abstract— **This work describes a high-speed simulation technique of analog circuits which is based on the use of state-space equations and an explicit integration method parallelised on a multiprocessor architecture. The integration step of such method is smaller than the one required by an implicit simulation technique based on Newton–Raphson iterations. However, given that explicit methods do not require the computation of time-consuming matrix factorizations, the overall simulation time is reduced. The technique described in this work has been implemented on a NVIDIA general purpose GPU and has been tested simulating the Gaussian filtering operation performed by a smart CMOS image sensor. Such devices are used to perform computation on the edge and include built-in image processing functions. Among those, the Gaussian filtering is one of the most common functions, since it is a basic task for early vision processing. These smart sensors are increasingly complex and hence the time required to simulate them during their design cycle is also larger and larger. From a certain imager size, the proposed simulation method yields simulation times two order of magnitude faster that an implicit method based tool such us SPICE.**

Keywords—simulation acceleration; state-space technique; nany-core; GPU; CMOS imager.

I. Introduction

Design of CMOS image sensors is certainly a complex task, even more difficult if such devices include built-in processing operators which allow them to operate under the edge-computing paradigm. Although digital processing is more powerful than analog one, analog functions implemented at pixel level require significantly less power consumption, an increasingly more important specification for wireless devices. Among the set of analog operators used for early vision tasks, Gaussian filtering is one of the more important functions. It is used to reduce the noise associated to the image capture without affecting subsequent processing stages. An analog implementation of this function can be achieved using RC networks, which is easily synthesisable in CMOS technology at pixel level of MOS transistors working on the triode region [1]. This implementation allows to control the channel resistance through the devices gate voltage, which in turns regulates the smoothing degree of the image. This illustrates the increasing complexity of current smart image sensors, whose analog design requires extremely large transient simulation times to evaluate the device performance. In order to reduce the time-to-market, the acceleration of such simulations becomes a keystone to improve the design productivity.

SPICE type simulators are widely used by current electronic design tools to perform transient simulations of electronic circuits. These simulators are based on the modified nodal analysis, which uses implicit differentiation techniques based on Newton–Raphson iterations to solve the analog equations at each time step. Although this numerical method has largely proven to be reliable and numerically stable, it consumes large CPU times which easily last hours or even days. As alternative to such methods, in [2] an explicit integration technique is used to simulate a mixed signal system modelled using state-space equations. The main drawback is that, compared to implicit integration techniques, explicit methods require smaller time steps in order to assure numerical stability. But given that they do not require computationally costly matrices calculations, their overall computational work load is lighter and so, the overall simulation time can be decreased.

However, simulation speed can be further improved using parallelisation techniques on many-core computers. Among those, general purpose Graphic Processing Units (GPUs) have been widely used in the last years in different engineering disciplines and also in the field of electronics to speed up the simulation of systems and circuits, in part thanks to the so called Compute Unified Device Architecture (CUDA) [3]. There are in the literature some works which have tested the acceleration of analog circuits simulation [4-6] while lately, specific works have focused on sparse matrix solvers by LU factorization [7-10], being these works still based on classical implicit integration methods. In this work, a numerical method based on an explicit integration schema parallelised on a general purpose GPU and able to speed up the simulation of passive analog circuits is illustrated. The method is applied to simulate a Gaussian filtering function implemented at pixel level on a CMOS image sensor. The rest of this paper is organized as follows: Section II introduces the linearized state-space technique and its

This work has been partially funded by Spanish government through project RTI2018-097088-B-C33 (MINECO/FEDER, UE) and by EPSRC (the UK Engineering and Physical Sciences Research Council) under grant EP/N0317681/1. The research stay at University of Southampton (UK) has been supported by Ministerio de Educación, Cultura y Deporte within the “Programa Estatal de Promoción del Talento y su Empleabilidad en I+D+i, Subprograma Estatal de Movilidad, del Plan Estatal de I+D+I” under grant PRX18/00565.

978-3-9819263-4-7/DATE20/©2020 EDAA

parallelisation on a general purpose GPU. Section III demonstrates the technique with an example of a filtering function implemented on a CMOS imager. The results obtained are presented in Section IV. Finally, conclusions and future work are discussed in Section V.

II. EXPLICIT INTEGRATION TECHNIQUE

Let (1) be the general state equation of a passive, nonlinear system:

$$\dot{x}(t) = f(x_t.t); x(0) = x_0 \qquad (1)$$

Its corresponding linearized state equation at time point t_k, for k = 0, 1…:. is defined as:

$$\dot{x}(t_k) = J_k x(t_k) + E e_x(t_k) \qquad (2)$$

where $x(t_k)$ is the vector of state variables at time t_k, e_x is the vector of excitations, J is the Jacobian matrix of the linearized model at time point t_k and E is a coefficient matrix for the excitations. For this system, its Adams–Bashforth integration scheme is described by:

$$x_{k+1} = x_k + (h \cdot \beta_0 \cdot J)x_k + h \cdot J \cdot \sum_{i=1}^{p} \beta_i x_{k-i}; k = 1 \dots \qquad (3)$$

where h is the time step and β_i. $i = 0, \dots. p$ are the Adams-Bashforth coefficients of an integration method of order p [11]. In an explicit integration method, the step size must be limited not only to bound the accuracy of the numerical solution, but also to guarantee the integration stability. For a second order method, its stability is achieved if $h\cdot||J|| <1$. Given that the calculation of $||J||$ is computationally expensive, an alternative method to estimate the maximum step size is presented in [3]. However, this method is valid for definite negative matrices, as shown in [12], a condition which is not always accomplished. Thus, in this work the following estimation is proposed. Given that the Euclidean norm of an m x n matrix, $||J||_F$ defined as:

$$||J||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|^2} \qquad (4)$$

is such that $||J||_F \geq ||J||$, then the step size value which guarantees stability is bounded by:

$$h \leq \frac{1}{\sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|^2}} \qquad (5)$$

Although the step sizes obtained using this technique are smaller than those used by an implicit integration method and so, a larger integration time is expected, the implicit method requires computationally costly matrices factorization calculations. Hence, the balance between these two computational tasks is expected to be favourable to the explicit method. The transient simulation of the described system is performed looping (2) and (3) along time. This sequence can be easily parallelisable on a many-core computer, since at each time step the calculation of the variables $\dot{x}$ only requires the previous values of x in an explicit method. So, for a given problem with N state variables, the algorithm can be run on N parallel processing units, each one of them working out the value of a single variable $\dot{x}_i$.

Algorithm 1: Parallelised integration scheme.

```
t=0                          // Initialization
do                           // Loops for simulation time
    GPU: begin               // Computes $\dot{x}_{i,k}$
        $\dot{x}_{i,k} = E_j \cdot e_{xk}$                          // (2)
        j=0;
        do
            $\dot{x}_{i,k} = \dot{x}_{i,k} + x_{i,j,k} J_{i,j}$     // (2)
            j++;
        while (j < N)
    GPU: end                 // Threads are synchronised
    GPU: begin               // Computes $x_{i,k+1}$
        $x_{i,k+1} = x_{i,k} + h \sum_{l=1}^{p} \beta_l x_{i,k-l}$  // (3)
    GPU: end
    k++;
    t=t+h;                   // Updates time
while (t < simulation time);
```

In this work, a general purpose GPU together with CUDA programming model has been used. The GPU used is able to run parallel threads which are grouped into warps and these into thread blocks. This architecture presents some characteristic to be taken into account when programming the device. On the one hand, all the threads within the same block can access a common shared memory, which is faster that the global memory accessed by threads from different blocks. On the other, synchronisation of threads inside of a same thread block is easier that performing the synchronisation of threads spread into different blocks. However, the number of threads per block is limited, and so a balance between number of threads and synchronisation speed must be reached. Moreover, given that threads inside a same warp are run following a single-instruction-multiple-threads (SIMT) execution model, the divergences of instructions between threads in a same warp forces that threads corresponding to different instructions are executed serially penalising the GPU efficiency. Finally, data transfers between CPU and GPU should be reduced to the minimum because the interaction between the CPU and the GPU a highly computational resources consuming process [13].

Algorithm 1 describes the parallel implementation of the explicit integration method on a general purpose GPU. The algorithm has been coded so that each thread in the GPU computes the value of a single state variable x_i. In each time step, the GPU kernel is run twice. The reason is that the computation of eq. (3) must wait until all the parallel threads have finished to compute the value of variables $\dot{x}$ according to eq. (2), which is done during the first GPU execution. Thus, this provides a mechanism to synchronise these two processes when the number of threads is such that several thread blocks are used.

III. ANALISYS OF THE CMOS GAUSSIAN FUNCTION

A CMOS image sensor is fabricated using standard CMOS technologies. It is composed by an array of pixels, each one containing a photodiode and between 1 to 4 transistors. Fig. 1 shows an m x n CMOS imager where the capacitors model the pixels storing the captured light intensity value as in a real CMOS imager. The MOS devices shown in the figure add the capability to perform an analog time-controlled Gaussian filtering on the image captured by the imager [1].

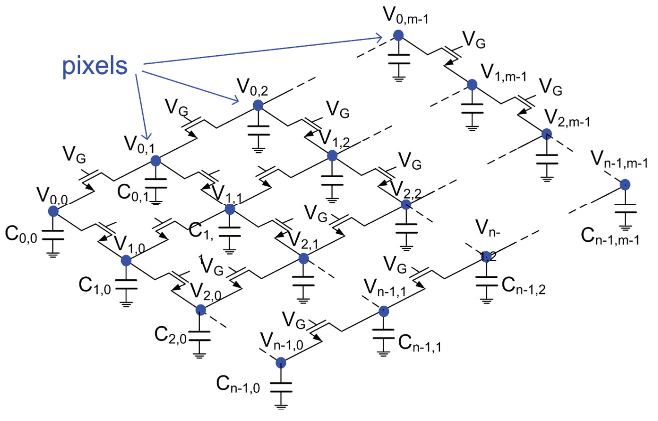


Fig. 1. Analog implementation of a Gaussian filtering MOS-C network.

The filtering function is implemented when the MOS transistors and the capacitors operate as an RC network. To this end, a common voltage V_G is applied to the gate of all transistors. When the image is being captured by the sensor circuitry (not shown in the figure), V_G is null. Once the image has been captured, the filtering operation starts by applying a suitable V_G to the transistor gates so that they operate in the triode region. Under these conditions, for a given pixel *(i,j),* the dynamics of the circuit is given by the following equation:

$$C\frac{dV_{i,j}}{dt} = -\frac{V_{i,j}-V_{i,j+1}}{R_{i,j+1}} - \frac{V_{i,j}-V_{i,j-1}}{R_{i,j-1}} - \frac{V_{i,j}-V_{i+1,j}}{R_{i+1,j}} - \frac{V_{i,j}-V_{i-1,j}}{R_{i-1,j}} \quad (6)$$

where $V\equiv V(t)$ and $R\equiv R(v)$. $R_{i,j+1}$ is the resistance of the MOS channel between nodes *(i,j)* and *(i,j+1)*. Given that the channel resistance depends on the drain to source voltage of the MOS device, it will vary with the capacitor voltage and thus, a more precise behavioural description is required. However, as shown in [1], due to the charge conservation property, the channel resistance of a MOS device connecting two capacitors $C_{i,j}$ and $C_{i,j+1}$ is given by:

$$R_{i,j+1} = \frac{L}{KW\left(2V_{GT}-\left(V_{i,j}(0)+V_{i,j+1}(0)\right)\right)} \quad (7)$$

being $V_{GT} = V_G\text{-}V_{th}$. and $v_k(0)$ the initial voltage of capacitor k. Equation (6) can be written as:

$$R_{i,j+1} = \frac{K_R}{\left(2V_{GT}-\left(V_{i,j}(0)+V_{i,j+1}(0)\right)\right)} \quad (8)$$

where K_R*=L/KW.* Let the voltage at each node be the vector of state variables. Using (8) in (6), the state equation of a capacitor voltage is given by:

$$\begin{gathered} K_R C\frac{dV_{i,j}}{dt} = \\ -\left(8V_{GT}+4V_{i,j(0)} + V_{i,j+1(0)}+V_{i,j-1(0)}+V_{i+1,j(0)}+V_{i-1,j(0)}\right)V_{i,j} \\ +\left(2V_{GT}+V_{i,j(0)} + V_{i,j+1(0)}\right) V_{i,j+1} \\ +\left(2V_{GT}+V_{i,j(0)}+V_{i,j-1(0)}\right) V_{i,j-1} \\ +\left(2V_{GT}+V_{i,j(0)} + V_{i+1,j(0)}\right) V_{i+1,j} \\ +\left(2V_{GT}+V_{i,j(0)} + V_{i-1,j(0)}\right) V_{i-1,j} \quad (9) \end{gathered}$$

Given that the number of neighbour pixels for the pixels placed on the edges and on the corner on the imager is respectively 3 and 2, instead of 4, eq. (9) is conveniently adjusted to model their behaviour. So, the set of equations which describe the evolution of the state variables $v_{i,j}$ is then:

$$\frac{d}{dt}\begin{pmatrix} v_{0,0} \\ v_{0,1} \\ \vdots \\ v_{m-1,n-1} \end{pmatrix} = \begin{pmatrix} A_{0,0} & A_{0,1} & 0 & \cdots & 0 \\ A_{10} & A_{1,1} & A_{1,2} & \cdots & 0 \\ 0 & A_{2,1} & A_{2,2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A_{n-1,n-1} \end{pmatrix}\begin{pmatrix} v_{0,0} \\ v_{0,1} \\ \vdots \\ v_{m-1,n-1} \end{pmatrix} \quad (10)$$

where $A_{r,r}$, are m *x* m submatrices defined as:

$$A_{r,r} = \begin{pmatrix} J_{nr,nr} & J_{nr,nr+1} & 0 & \cdots & 0 \\ J_{nr+1,nr} & J_{nr+1,nr+1} & J_{nr+1,nr+2} & \cdots & 0 \\ 0 & J_{nr+2,nr+1} & J_{nr+2,nr+2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & J_{nr+m-2,nr+m-1} \\ 0 & 0 & 0 & \cdots & J_{nr+m-1,mr+m-1} \end{pmatrix} \quad (11)$$

and $A_{r,s}$, for $r\neq s$, are m *x* m submatrices defined as:

$$A_{r,s} = \begin{pmatrix} J_{nr,ns} & 0 & \cdots & 0 \\ 0 & J_{nr+1,ns+1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & J_{nr+m-1,ns+m-1} \end{pmatrix} \quad (12)$$

In both equations (11) and (12), $J_{i,j}$ are the coefficients obtained from applying eq. (9). Since all the parameters in (9) are constant, these coefficients can be computed before the integration loop begins. Moreover, the maximum step size that guarantees stability, can be also computed using (5) before the simulation is executed. According to [14], the variation of the voltage along time is a Gaussian function with $\sigma = (2t_{ON}/RC)^{0.5}$, where t_{ON} the time during which the MOS devices are operating in the triode region.

IV. Results

The simulation technique described in previous section has been applied to simulate the system described in equations (10) to (12). To test the speedup of the proposed method, several transient simulations of 1μs each have been run for CMOS imagers of different sizes. The many-core processor used has been a NVIDIA GeForce GTX 1080, 3584 Core, 1531MHz and 11 GB of RAM GPU. To evaluate the speed up of the proposed technique, an RC and a MOS-C model of the imager have been also simulated using Ngspice on an AMD Ryzen Threadripper 1950X 16-Core Processor, 2180 MHz and 64 GB of RAM. All the simulations have been done modelling the capacity of the CMOS imager pixels as C=10 pF and using the following MOS transistor parameters: K=50 μA/V^2, W/L=1, V_{th}=0.5 V and V_G=3 V.

TABLE I. SIMULATION TIMES FOR EXPLICIT AND IMPLICIT METHODS

Image size	*Explicit on GPU (s)*	*Ngspice on CPU (s)*	
		RC	***MOS-C***
8 x 8	0.0003	0.005376	0.010411
16 x 16	0.0007	0.02822	0.042331
32 x 32	0.008	0.2317	0.348719
64 x 64	0.040	3.61	4.28152
128 x 96 (SQCIF)	0.230	34.6908	42.8304
128 x 128	0.434	82.3172	88.021
176 x 120 (QCIF)	1.103	138.521	144.144

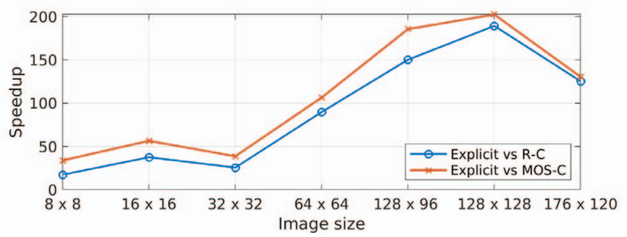
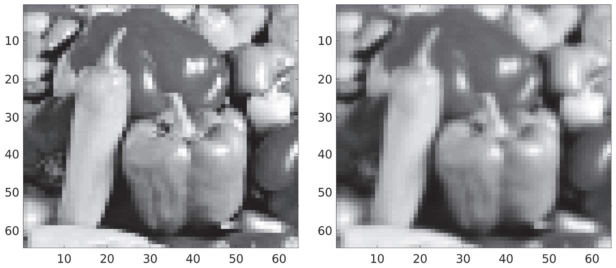


Fig. 2. Speedup of the proposed method vs Ngspice for an RC network and for a MOS-C network.



a) b)

Fig. 3. Transient simulation of the MOS-C image Gaussian filter applied to a 64 x 64 pixels image for t=10ns. a) Original image b) Filtered image

The processor time required for each transient simulation is detailed in Table I. As expected, the Ngspice simulation time required for the MOS-C model is larger than for the RC model and in both cases, the simulation time is larger than that achieved using the proposed explicit method. Moreover, the speedup achieved is above two orders of magnitude for imagers larger than 64x64 pixels, as shown in Fig. 2. This speedup is however penalised for the QCIF image format due to the large number of threads used by the GPU which requires the use of threads from different blocks. So, the data transfer between different thread blocks is a drawback for the implementation on a GPU. Fig. 3 shows an example of the simulation of a 64 x 64 pixels image filtering using the described method for a MOS conduction time of 10 ns. Compared to the simulation using Ngspice, the explicit one presents an absolute root mean square error (RMSE) of 4.6e-3 units.

V. CONCLUSIONS AND FUTURE WORK

This work has shown a technique to reduce the transient simulation time of a passive analog circuit such a CMOS image sensor with a built-in filtering function. Although the method has been demonstrated for this particular circuit, it can be used with other type of analog circuits. Depending on the circuit characteristics, the technique to estimate the maximum step size which guarantees stability to the numerical solution can be even simplified. The described technique has shown a speedup of two orders of magnitude with respect to an implicit integration method running on a CPU. One of the limitations of this technique is due to the GPU architecture, which limits the speed of the algorithm when large amount of data is transferred between threads placed in different thread blocks. On the other hand, the improvements can come with the implementation of sparse matrices multiplication techniques and with the use of a variable step implementation.

REFERENCES

[1] J. Fernandez-Berni and R. Carmona-Galan. “All-MOS implementation of RC networks for time-controlled Gaussian spatial filtering”. Int. J. Circ. Theor. Appl. 2012; vol. 40. pp. 859–876

[2] T. J. Kazmierski. L. Wang. B. M. Al-Hashimi and G. V. Merrett. "An Explicit Linearized State-Space Technique for Accelerated Simulation of Electromagnetic Vibration Energy Harvesters." IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems. vol. 31. pp. 522-531. April 2012.

[3] NVIDIA. CUDA C Programming Guide Version 7.0. Accessed: Mar. 5, 2018. [Online]. Available: http://docs.nvidia.com/cuda/cudac-programming-guide/

[4] K. Gulati, J. F. Croix, S. P. Khatri and R. Shastry, "Fast circuit simulation on graphics processing units," *Asia and South Pacific Design Automation Conference*, Yokohama, 2009, pp. 403-408.

[5] R. E. Poore, "GPU-accelerated time-domain circuit simulation" *IEEE Custom Integrated Circuits Conference*, Rome, 2009, pp. 629-632.

[6] L. Han and Z. Feng, "TinySPICE Plus: Scaling up statistical SPICE simulations on GPU leveraging shared-memory based sparse matrix solution techniques," IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, 2016, pp. 1-6.

[7] X. Chen, L. Ren, Y. Wang and H. Yang, "GPU-Accelerated Sparse LU Factorization for Circuit Simulation with Performance Modeling" *IEEE Trans. on Parallel and Dist. Systems*, vol. 26, pp. 786-795, March 2015.

[8] O. Schenk and K. Gartner, “Solving unsymmetric sparse systems of linear equations with PARDISO,” Future Generat. Comput. Syst., vol. 20, no. 3, pp. 475–487, Apr. 2004.

[9] W. Lee, R. Achar and M. S. Nakhla, "Dynamic GPU Parallel Sparse LU Factorization for Fast Circuit Simulation," *IEEE Transactions on Very Large Scale Integration Systems*. doi: 10.1109/TVLSI.2018.2858014.

[10] T. A. Davis and E. P. Natarajan, “Algorithm 907: KLU, a direct sparse solver for circuit simulation problems,” ACM Trans. Math. Softw., vol. 37, no. 3, Sep. 2010, Art. no. 36

[11] D. Zwillinger, Handbook of Differential Equations, 2nd ed. San Diego, CA: Academic, 1989.

[12] G. Domenech-Asensi and T. J. Kazmierski ”An efficient numerical solution technique for VLSI interconnect equations on many-core processors”, in Proc. Intl. Symp. on Circ. and Systems, Sapporo, Japan, 2019.

[13] CUDA C best practices guide, November 2019,. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf

[14] B.E. Jahne “Multiresolutional signal representation”. In Chapter 4, Handbook of Computer Vision and Applications. Volume 2: Signal Processing and Pattern Recognition, Academic Press, 1999.