Stability and efficiency of explicit integration in interconnect analysis on GPUs

Gines Domenech-Asensi (1) and Tom J. Kazmierski (2)

(1) Departmento de Electronica y Tec. de Computadoras, Universidad Politecnica de Cartagena, Cartagena, 30201, Spain, e-mail: gines.domenech@upct.es
(2) Department of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, United Kingdom, e-mail: tjk@ecs.soton.ac.uk

***Abstract*—This paper presents a technique to parallelise a numeric integration solver on general purpose GPU. The technique is based on the combination of space state modeling with an explicit integration method based on the Adams-Bashforth second order formula. The paper studies the stability of variable step explicit method and proposes a technique to guarantee integration stability using this technique. Although explicit methods require smaller integration steps compared to the traditional implicit techniques, they avoid the complex calculations on large which are used to solve the last ones. The technique is demonstrated simulating an RC model of an VLSI interconnect. Results achieved by the proposed variable step explicit method is compared to those achieved by a traditional implicit integration based simulator like Ngspice. The results show that the parallelised explicit solution is one order of magnitude faster than the implicit one for increasingly complex circuits.**

***Index Terms*—Simulation acceleration, state-space technique, many-core computer, GPU.**

I. Introduction

During the last four decades, the cost of integrated circuits has been kept minimal by doubling component density every year, while improving the power consumption and performance of the devices. However, the interconnect performance has been continuously degraded being the wire resistance a dominant factor of this behaviour. Given that delays caused by RC parasitics can easily spoil the improvements reached by new device and circuit architectures in the context of a continuous evolution of technology nodes, careful analysis of the behaviour of such elements in the early stages of a system design are needed to achieve the desired circuits performance.

These analysis require, nevertheless, transient simulations which consume enormous amounts of time using traditional circuit simulators. Classical simulators like SPICE rely on the modified nodal analysis and use implicit integration techniques based on Newton—Raphson method to solve the circuit analog equations at each time step. These methods have proven to be reliable and numerically stable, but on the other hand, they lead to long CPU times, often hours or even days and weeks, which contribute to extend the design cycle time. The main reason for this extensive computation time is due to computations required to factorize the jacobian matrix of the analog system. Unlike these methods, the computational workload of explicit integration techniques is lighter, and although they require significantly smaller time steps compared to implicit methods, their overall computation time is smaller. Different works have proved that the use of state-space equations combined with explicit integration methods is a suitable technique to speed up transient simulations of analog circuits [1] or mixed systems [2]. However, given the increasing complexity of analog circuits and systems, new techniques are required to speed transients simulations, besides the use of alternative integration algorithms. Among these techniques, those based on exploiting the parallelization of analog integration methods running on parallel computer architectures are arising in the last years. The so called Compute Unified Device Architecture (CUDA) [3] in 2006, is a programming model that allows engineers to use a high level programming language such as C to develop algorithms for general purpose Graphics Processing Units (GPUs). This has given access to relatively cheap parallel architecture computers to many engineers which now can perform fast simulations of different types of scientific computations. Thus, in the last decade, there have been different proposals to accelerate the simulation of analog circuits using GPUs [4]–[7]. Some works have focused on sparse matrix solvers [8] or LU factorization matrix solver [9]–[11], which have achieved different speedups compared with parallel sparse solvers like PARDISO [12] or KLU [13]. NVIDIA also released an official sparse matrix solver, cuSolver [14], but the LU factorization in it is still performed in CPU instead of GPU.

A common characteristic of these works is that they are still focused on the traditional implicit integration methods used for simulators like SPICE. Recently, an explicit integration method parallelizable over a many-core processors has been proposed [15]. This method combines space state equations with a fixed-step explicit schema to speedups the simulation of passive circuits of a complexity up to 1000 nodes. Nevertheless, this technique can be improved employing variable step simulations, in a similar fashion that traditional implicit simulators, which would lead into more accurate simulations. To apply this technique, a stability analysis is depth is required to bound the value of the time step. Up to our knowledge, there is not any work related to interconnect simulation using

this variable-step method. Thus, in this paper, a variable-step explicit integration schema parallelizable over a many-core processors, is proposed. The proposed method evaluates variable step techniques running on a general purpose GPU. The technique is validated performing extensive transient simulations of large VLSI interconnects. The results are compared to those achieved by the Ngspice simulator [16] running on a multiprocessor GPU. The rest of the paper is organised as follows. Section II describes the linearized space state integration technique and its implementation on a many-core computer. Section III shows an example of the proposed technique. Finally, conclusions are drawn up in Section IV.

II. Linearized Space State Technique

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

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

Its linearised state equation at time point $t_k, k = 0, 1 \ldots$ is given by:

$$\dot{X}(t_k) = J_k X(t_k) + E e_x \tag{2}$$

being X the vector of N state variable wave-forms, e_x a vector of excitations and J_k and E coefficient matrices, where J_k is the Jacobian of the linearized model at the time point t_k. The eigenvalues of J_k have negative real parts given that the system is passive [17]. Currently, implicit Newton-Raphson based integration schemas are used by most circuit simulators, since they assure numerical stability. However, explicit methods can be used to provide a fast integration process for linearized state equations if the step-size is limited to assure stability besides controlling the accuracy of the numerical solution. This limit can be set without consuming much CPU time using the stability technique described in [2], which takes advantage of the system's passivity and uses a fast method for estimating the maximum allowed step size directly from the Jacobian entries. However, this method is restricted to systems with a diagonal dominant Jacobian Matrix and it must be reformuled to be useful for other types of Jacobians.

A. Stability analysis

Fig. 1 shows a finite difference grid for a q-order Adams-Bashforth method, where t_k is the current time point and t_{k+1} is the next time point. I represents the integration interval and $P_q(t)$ is the interpolation polynomial.

Let $h_i = t_{i+1} - t_i$ be the time step between two consecutive time points t_i and t_{i+1}. In a fixed step integration method, all the h_i values are equal and invariable on time. However, in a variable step method, the values of h_i are different, and change with time. The expressions for the general variable-step method can be obtained by integrating the divided difference polynomial approximation between the current variable value $x_k \equiv x(t_k)$ and the predicted one $x_{k+1} \equiv x(t_{k+1})$.

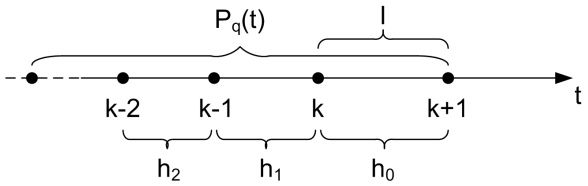

Fig. 1. Finite difference grid for the q^{th}-order Adams-Bashforth method.

TABLE I
Maximum real negative value of the stability plots

r	1	1.2	1.4	1.6	1.8	2
$L_r(h_0)$	-1	-0.952	-0.909	-0.869	-0.833	-0.8
$L_r(h_1)$	-1	-0.794	-0.649	-0.544	-0.463	-0.4

$$\begin{aligned} I = \int_{x_k}^{x_{k+1}} dy = \int_{t_k}^{t_{k+1}} P_q(t_{k+1})dt \Rightarrow \\ \Rightarrow x_{k+1} - x_k = \int_{t_k}^{t_{k+1}} \Big(f_0 + (x - x_k) f_k^{(1)} + ... \\ ... + (x - x_k)...(x - x_{k-p}) f_k^{(q)} \Big) dt + O \end{aligned} \tag{3}$$

where $f_k^{(q)}$ is the q^{th} divided difference of function f at t_k [18] and O is the truncation error. So, for the second order method, the state variable at time t_{k+1} is computed as:

$$x_{k+1} - x_k = f_k h_0 \left(1 + \frac{h_0}{2h_1}\right) - f_{k-1} h_0 \left(\frac{h_0}{2h_1}\right) \tag{4}$$

being $f_k = f_k^{(0)}$. For the third order method, the following term is added to (4):

$$\begin{aligned} \Bigg(\frac{t_{k+1}^3 - t_k^3}{3} - (t_k + t_{k-1}) \frac{t_{k+1}^2 - t_k^2}{2} \\ + t_k t_{k-1} (t_{k+1} - t_k) \Bigg) f_k^{(2)} \end{aligned} \tag{5}$$

Fig. 2 shows the stability plots of equations (4) and (5) for different ratios from 1 to 2 between the integration step sizes. The plots show how the variation of the ratio $r = h_{i+1}/h_i$ affects the stability for both the second and the third order AB methods. The values of integration step h_i decrease as the ratio is increased, being clearly smaller for the third order method. So, in order to be able to manage larger values of h_i, in this work the second order variable step integration method has been used. Table I shows the intersection (L_r) of the stability plots with the negative real axis for different values of r for the second order method.

For a system of linearised equations, substituting the values of f_k and f_{k+1} in (4) by the product $J_k X(t_k)$ as in (2), yields:

$$X_{k+1} = X_k + h_0 \beta_0 J \cdot X_k - h_0 \beta_1 J \cdot X_{k-1} \tag{6}$$

being $\beta_0 = 1/r + 1/2r^2$ and $\beta_1 = 1/2r^2$. According to the plots in Fig. 2, the stability is achieved if the product $h \cdot ||J||$ is within the left hand complex plane and bounded by the

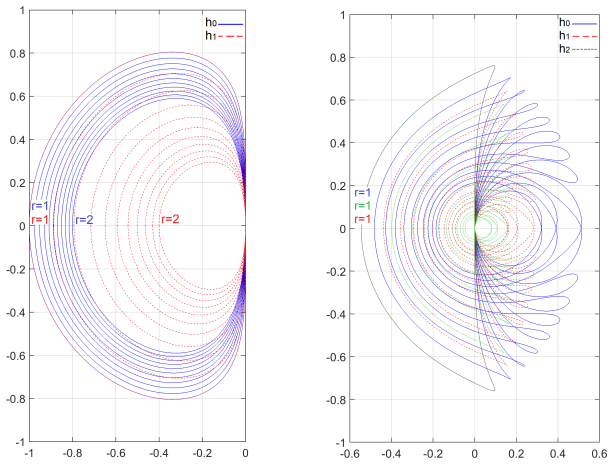

Fig. 2. Stability regions for the second and third order AB methods.

stability curve. For a symmetric Jacobian, $\max_{i=1,\ldots,N} \lambda_i = ||J||$ and according to the Gershgorin theorem the values of the eigenvalues are bounded to $\lambda \leq J_{ii} + \sum_{j=1}^{N} |A_{ij}|$ for $i \neq j$. Given that all the Jacobian entries are real numbers, we finally obtain that:

$$h \leq \frac{L_r}{\sum_{j=1}^{N} J_{ij}} \tag{7}$$

being L_r the intersection of the stability plot for a given step-ratio r with the negative semi-axis of the complex plane.

Step sizes obtained from using this technique are expected to be smaller than the maximum step sizes used in implicit methods, which are stable and hence, the step sizes are used to define the accuracy. However, the advantage of this technique is speed, given that time-consuming matrices factorization calculations in implicit methods are avoided.

B. Parallel implementation

The linearised space state system of equations described by (2) can be computed in a parallel architecture at each time point t_k, given that each state variable can be worked out at each time point independently of the rest of the state variables.

Algorithm shown in Fig.3 describes the procedure to compute the values of the state variables on a general purpose. Compared to the fixed step implementation [15], the proposed variable step algorithm presents the following differences. First, there are two calls to GPU parallel execution in each time step. The first call is launched to compute the incremental values of the state variables, and to compare these values with the specified tolerances. The second call is run first to determine if the tolerances have been violated and second to update the values both of the state variables and the time steps h.

These two calls to the parallel execution are required to provide a synchronization mechanism before evaluating if there has been a violation of tolerance in any GPU thread.

1: $t \leftarrow 0$
2: **while** $t \leq SimTime$ **do**
3: **GPU: begin** ▷ 1^{st} **call many-core processor**
4: $\dot{x}_{i,k} \leftarrow E_j e_{x,k}$
5: $j \leftarrow 0$
6: **while** $j \leq N$ **do** ▷ **Compute (2)**
7: $\dot{x}_{i,k} \leftarrow \dot{x}_{i,k} + x_{i,j,k} J_{i,j}$
8: $j \leftarrow j + 1$
9: **end while**
10: $\Delta x_{i,m} \leftarrow h_m \sum_{l=1}^{p} \beta_l \dot{x}_{i,k-l}; k = 1, ...m \in [1, 3]$
11: **if** $\Delta x_{i,m} > tolerance$ **then** ▷ **Check tolerance**
12: $ctrl\_reg[m] \leftarrow 1$
13: **end if**
14: **GPU: end** ▷ **Synchronization of all threads**
15: **GPU: begin** ▷ 2^{nd} **call many-core processor**
16: $update \leftarrow 0$
17: **if** $ctrl\_reg[1] = 0$ **then** ▷ **If tolerance is ok**
18: **if** $ctrl\_reg[2] = 0$ **then** ▷ **If bigger step is ok**
19: $x \leftarrow x + \Delta x_{i,2}$ ▷ **update** x
20: $h_m \leftarrow r h_m; m \in [1, 3]$ ▷ **and increase step**
21: **else**
22: $x \leftarrow x + \Delta x_{i,1}$ ▷ **else, update** x
23: **end if**
24: $update \leftarrow 1$
25: **else** ▷ **If tolerance is not ok**
26: **if** $ctrl\_reg[0] = 0$ **then** ▷ **but smaller step is ok**
27: $x \leftarrow x + \Delta x_{i,0}$ ▷ **update** x
28: $h_m \leftarrow h_m / r; m \in [1, 3]$ ▷ **and decrease step**
29: $update \leftarrow 1$
30: **else**
31: $h_m \leftarrow h_m / r^2; m \in [1, 3]$ ▷ **else decrease step**
32: **end if**
33: **end if**
34: **GPU: end**
35: **if** $update = 1$ **then** ▷ **If** Δx_i **is ok**
36: $t \leftarrow t + h_1$ ▷ **updates time and timestep**
37: $k \leftarrow k + 1$
38: **end if**
39: **end while**

Fig. 3. Variable step parallelised integration method.

Depending on the many–core architecture, a single parallel execution can be done, using a synchronization call at this point instead. A second difference is that there is a step control routine running inside the second parallel execution call. The step size control is as follows. In each iteration step, three values of Δx_i are computed: $\Delta x_{i,1}$ for the current step size, $\Delta x_{i,2}$ for a higher step size, and $\Delta x_{i,0}$ for a lower one. The three values of Δx_i obtained ($\Delta x_{i,0}$, $\Delta x_{i,1}$, $\Delta x_{i,2}$) are then compared with a predefined tolerance values to determine which increment must be added to state variables value at the current step time. The result of the comparison is annotated in the control register. In the second parallel execution call, if the value of $\Delta x_{i,1}$ is within the limits of the specified tolerance,

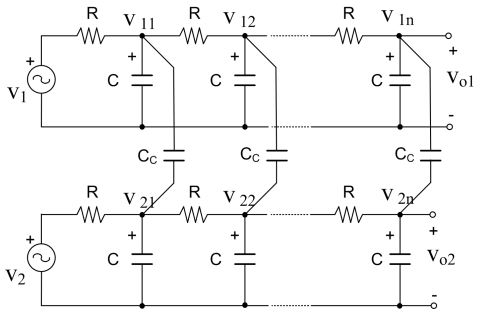

Fig. 4. RC model of the coupled interconnect.

the three step values are increased an amount Δh and $\Delta x_{i,1}$ is used to update the state variable x_i aeig(t each many-core parallel process, being the new values the current ones multiplied by a ratio r. However, if the value of $\Delta x_{i,0}$ is larger that the specified tolerance, then a lower step h is required. In this case, the value of $\Delta x_{i,2}$ is checked to determine the rate of decrease of the integration step, r for a slow decrease or r^2 for a faster one. If the value of $\Delta x_{i,2}$ is lower than the tolerance, then it can be used to update the state variables. Otherwise, the values of of Δx_i computed in the current iteration step cannot be used. This provides a fast mechanism to adapt the rate of decrease of h to the rate of change of the system variables and to reduce the number of parallel executions.

III. Example

Fig. 4 shows the RC model of two coupled VLSI interconnect. In the figure, each interconnect is excited by a voltage source and is composed by n RC stages. Capacitors Cc represent the coupling between both interconnects. The space state equation of this model is given by:

$$\frac{RC(2C+C_c)}{C+C_c}\frac{d}{dt}\begin{bmatrix} v_{11} \\ \vdots \\ v_{1n} \\ v_{21} \\ \vdots \\ v_{2n} \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \begin{bmatrix} v_{11} \\ \vdots \\ v_{1n} \\ v_{21} \\ \vdots \\ v_{2n} \end{bmatrix} + \begin{bmatrix} v_{in} \\ \vdots \\ 0 \\ \frac{C_c}{C+C_c}v_{in} \\ \vdots \\ 0 \end{bmatrix} \tag{8}$$

being the submatrices $J_{11} = J_{22}$ equal to:

$$J_{11} = J_{22} = \begin{bmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \dots & 0 \\ 0 & 1 & \ddots & & \vdots \\ \vdots & \vdots & & -2 & 1 \\ 0 & 0 & \dots & 1 & -1 \end{bmatrix} \tag{9}$$

while the value of $J_{12} = J_{21}$ is given by:

$$J_{12} = J_{21} = \frac{C_c}{C+C_c} J_{11} \tag{10}$$

The performance of the proposed algorithm has been tested on a general purpose GPU. The one used is a NVIDIA

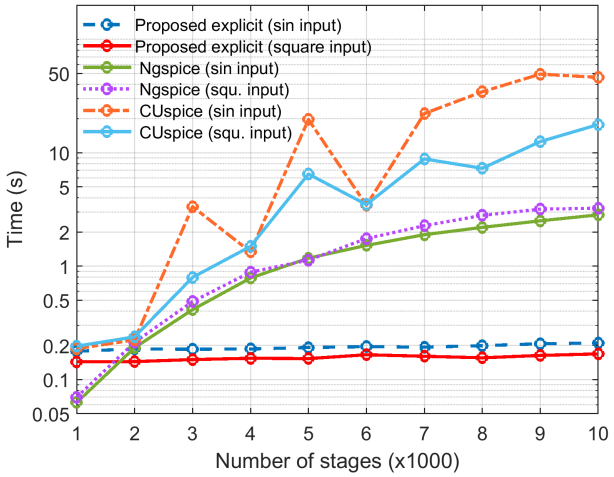

Fig. 5. Required simulation time for different number of stages.

GeForce GTX 1080, 3584 Core, 1531 MHz and 11 GB of RAM. The host processor is an AMD Ryzen Threadripper 1950X 16-Core Processor, 2180 MHz and 64 GB of RAM. Different transient simulations of $1\mu s$ each have been performed to measure the time required by the whole processing system, for different number of RC stages. Fig. 5 shows the time required by each simulation using the second order variable step explicit integration method. The ratio between h_0 and h_1 has been set to 1.05 after intensive tests with different input waveforms to evaluate the speed of the algorithm. In this example, the value of the passive components were R = 10 Ω, C = 250 pF and Cc = 2.5pF. So, according to (7) and table I, the maximum value of h which assures stability is 0.65 ns. Source v_2 was grounded and v_1 was 1V. The variable step approach was simulated using square and sine wave input signals. The initial time step was set to 1 ps.

The performance of the explicit integration technique is compared to that achieved by a well known variable-step implicit integration based software such as Ngspice running on the host processor and also to CUspice [19], an experimental extension of the former which runs on NVIDIA GPUs. The results show that for the smaller circuits circuits up to 2,000 RC nodes approximately, Ngspice proves to be faster. However, for larger circuits, the parallelised explicit method requires lower processing times, exceeding one order of magnitude for the largest circuits. The simulation time achieved by CUspice is considerably larger, mainly because its improvements are focused on the speedup the model evaluation process rather than in the matrix factorization. Also the parallelization algorithm seems not to be optimized for different number of threads, since the simulation times are not monotonically increased with the number of stages.

IV. Conclusion

This paper has presented a numeric integration method based on a variable-step explicit integration technique parallelised on a general purpose GPU. Up to our knowledge, this is the firs implementation of a variable step explicit integration

method used to evaluate interconnect. The paper has compared the proposed approach with a traditional implicit integration based simulators like Ngspice, being one order of magnitude faster.

Although this technique has been demonstrated using a GPU, it can be translated to other many-core computer architectures. Moreover, it provides a synchronization procedure between the calculation of the state variables derivative and the updating of both the integration step value and the state variable values which allow to simulate circuits of any size, being the only limitation the size of the computing architecture.

V. Acknowledgements

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 by Fundación Séneca-Agencia de Ciencia y Tecnología de la Región de Murcia, Programa Regional de Movilidad, Colaboración e Intercambio de Conocimiento Jimenez de la Espada under grant 21187/EE/19.

References

[1] K. C. A. Lam and M. Zwolinski. "Circuit simulation using state space equations" in Proc. 9th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME). Villach. 2013. pp. 177-180.

[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 Integ. Circ. and Syst. vol. 31, pp. 522-531. Apr. 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," in Proc. Asia and South Pacific Design Automation Conference, Yokohama, 2009, pp. 403-408.

[5] R. E. Poore, "GPU-accelerated time-domain circuit simulation" in Proc. 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," in Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, 2016, pp. 1-6.

[7] E. Schneider, M. A. Kochte, S. Holst, X. Wen and H. Wunderlich, "GPU-Accelerated Simulation of Small Delay Faults," IEEE Trans. on Comp.-Aided Design of Integ. Circ. and Syst., vol. 36, no. 5, pp. 829-841, May 2017.

[8] Y. Liang, W. T. Tang, R. Zhao, M. Lu, H. P. Huynh and R. S. M. Goh, "Scale-Free Sparse Matrix-Vector Multiplication on Many-Core Architectures," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 36, no. 12, pp. 2106-2119, Dec. 2017.

[9] 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 Distr. Syst., vol. 26, pp. 786-795, Mar. 2015.

[10] K. He, S. X. -. Tan, H. Wang and G. Shi, "GPU-Accelerated Parallel Sparse LU Factorization Method for Fast Circuit Analysis," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 24, pp. 1140-1150, Mar. 2016.

[11] 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, vol. 26, pp. 2518-2529, Nov. 2018.

[12] 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.

[13] 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.

[14] Cusolver Library, document DU-06709-001_v9.0, Jun. 2017.

[15] 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.

[16] Ngspice Circuit Simulator. Accessed: May. 24, 2019. [Online]. Available: www.ngspice.org

[17] L. O. Chua and P. Y. Lin. Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques. Englewood Cliffs. NJ: Prentice-Hall. 1975.

[18] J. D. Hoffman, Numerical Methods for Engineers and Sciencists 2nd Ed.,CRC Press, 2001.

[19] F. Lannutti, F. Menichelli, M. Olivieri "CUSPICE The revolutionary NGSPICE on CUDA Platforms", in Proc. 12th MOS-AK Workshop at the ESSDERC/ESSCIRC Conference, Venice, Italy, 2014.