GRIDIFICATION AND PARALLELIZATION OF ELECTRONIC CIRCUIT SIMULATOR

Marko Dimitrijević, Bojan Anđelković, Milan Savić, Vančo Litovski, *Faculty of Electronic Engineering Niš*

Abstract – *This paper presents the concept of gridification and parallelization of an electronic circuit mixed-mode simulator. Basic information regarding modern simulation, leading to the need for parallel simulation is presented. An overview of parallel simulation algorithms and implementations is given. Implementation of a new algorithm for parallel equation formulation in pAlecsis simulator is presented.*

1. INTRODUCTION

The development of low-cost personal computers with high computing power and gigabit LAN network connections in past decade, provided possibility for implementation of inexpensive distributed multiprocessor systems such as clusters (Fig. 1).

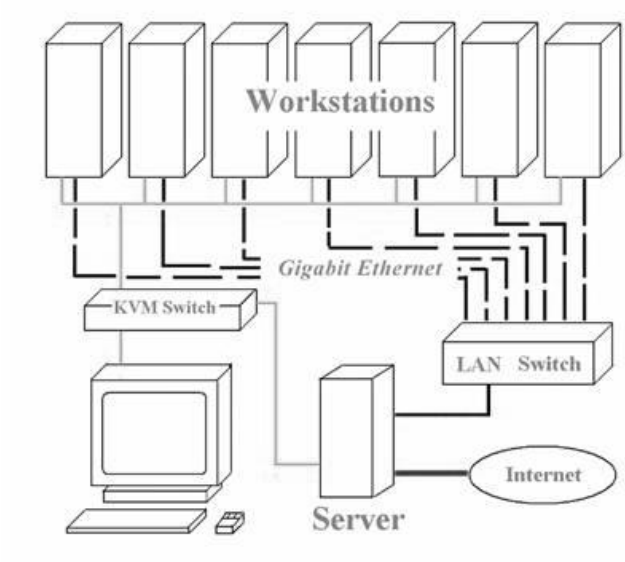

Fig. 1 Organization of the computer cluster

A cluster has many advantages over classic supercomputer: it is inexpensive, flexible, easy to use, easy for maintenance and highly stackable. The total price of computer cluster is more than ten times lower then dedicated supercomputer with similar computing power, and after amortization cluster nodes can be used as single personal computers. It usually uses open-source operating system, well documented and well known to programmers and system administrators. A cluster is also highly stackable: it can be easily extended by adding additional node or demoted by subtracting one. One particular version of this approach, involving open source system software and dedicated networks, has acquired the name "Beowulf" [1].

The development of Internet and WAN links of great capacity led to a new paradigm: the computational grid. The intention was to associate to the electrical power grid. In the same way electrical power could be obtained from power grid, computational power should be obtained *on demand* from a network of *providers*, potentially belonging to the entire Internet [2]. In the beginning, this paradigm has been strictly scientific and academic; but as the Internet, it became widely accepted and popular. One of the most common definitions says that a computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities providing on-demand access to computing, data, and services [3]. Basically, grid computing intends to provide access to resources using wide area connections; it can be determined as cooperation of geographically distributed computer systems (clusters) where user jobs can be executed.

The hardware part of computational grid infrastructure can be extremely heterogeneous. It consists of a number of clusters containing various number and types of processors, amounts of memory, LAN and WAN connectivity and mass-storage capacity.

The role of the software components is to provide distributed services for job submission and management, file transfer, database access, data management and monitoring. They also ensure security in multi-user environment using certificates. The software part consists of two layers: operating system and middleware.

Grid computing is suitable for intensive calculations that require significant processing power, large operating memory and throughput, as well as storage capacity. The simulations of integrated electronic circuits are paradigmatic example of these calculations.

This paper presents the concept of gridification and parallelization of an electronic circuit simulator.

With the rapid growth of electronic systems complexity, the simulation became a crucial step in design flow. Circuit simulation has proven to be one of the most important computer aided design (CAD) methods for the analysis and validation of integrated circuit design.

Today's high end integrated circuits contain both analog and digital components. In addition, non-electric elements are implemented within the electric integrated circuits, e.g. in microelectromechanical systems (MEMS). Complexity of modern electronic circuits has imposed the requirement for enabling the hardware designer to model a design at different levels of abstraction. All these trends have resulted in the development of the *mixed-mode simulation* domain. This paradigm includes mixed-signal (analog and digital), mixed domain (electrical and non-electrical) and mixed-level simulation. Obviously, this introduces unique difficulties for the modeler and simulation developer [3].

In order to simulate a system, it has to be described i.e. modeled. Mixed-mode systems are described using:

1. algebraic equations
2. ordinary differential equations (ODE)
3. partial differential equations (PDE)
4. algorithms
5. logic states

Algebraic equations describe resistive portion, whereas ODE describe dynamic portion of the system. ODEs are discretized in order to create sets of nonlinear algebraic equations.

Let us assume that PDEs describe mechanical part of the system. In order to obtain set of ordinary differential equations, space discretization of PDEs is performed, where the system unknowns are the spatial displacements as functions of time. Thus, PDEs introduce new sets of ODEs, whose number is large, depending on a discretization grid. [5]

Thus, one comes down to the problem of formulation and solution of systems of nonlinear algebraic equations, that are to be solved iteratively – with the help of linearization i.e. by application of Newton methods. Evaluation of all derivatives that are necessary for the linearization is the most time consuming part of the simulation process.

So far we have considered analog/continuous models. Algorithmic description may define behavior of continuous (analog, pulse…) as well as for discrete (logic expressions, tables, programs) models. This kind of model's behavior description needs to be translated into other kinds of description.

Discrete-event processes define the behavior of the discrete-event model described by the logic states. The simulation mechanism for discrete-event simulation differs from continuous time simulation. It involves the use of future event tables, containing the information of events to be processed.

In mixed-signal systems, a specific time advancement algorithm has to be implemented in order to synchronize events between analog and logic part. In addition, specific algorithms are to be implemented in order to convert signals at the digital – analog meeting points.

To conclude this brief overview of modern simulation techniques, we would like to emphasize that, according to the presented facts, simulation process may be characterized as memory intensive, computationally intensive and algorithmically complex. This leads to long simulation runtimes. Having in mind that every design needs many simulation runs of the same system in order to get optimal solutions with respect to many different requirements, it is obvious that long simulation runtimes lead to delay in design process. One possibility to reduce these runtimes is to divide the circuit into several partitions and to simulate the partitions in parallel.

Barriers to the widespread use of parallelism are in all three of the usual large subdivisions of computing: hardware, algorithms and software. As for the hardware, intercommunication networks that keep up with speed of advanced single processors are still not available. The biggest obstacle is inadequate software. Compilers that automatically parallelize sequential algorithms remain limited in their applicability. Best performance is still obtained when programmer himself supplies the parallel algorithm.

There are number of parallel computational models in use today, such as: data parallelism, shared memory, message-passing, remote memory operations, threads and various combined models.

In our research we focus on message passing model of parallel computation, and in particular the Message Passing Interface (MPI) instantiation of that model [6][7]. The message-passing model uses a set of processes that have only local memory but are able to communicate with other processes by sending and receiving messages. It is a defining feature of the message-passing model that data transfer from the local memory of one process to the local memory of another requires operations to be performed by both processes. MPI is being widely used and is expected to be around for a long time due to it's advantages over other models, which are: universality (it matches the hardware of most today's parallel supercomputers), expressivity (it is a useful and complete model in which to express parallel algorithms), ease of debugging, and performance. Message passing has become a standard for portability, in both syntax and semantic. The MPI standard [8, 9] was completed in 1997.

During the first phase of the simulator parallelization, we parallelized equation formulation for analog circuits described by nonlinear ODEs. Also, the appropriate algorithm for parallel solution of the system of linear equations will be included in the parallel simulator as the first next step in further development.

Improvements in parallel simulation of systems modeled by partial differential equations, parallel discrete-event as well as parallel mixed-signal simulation will be implemented in the future versions of the parallel simulator. Also, it will be enabled to perform parallel simulations on the computational grid.

2. CONTINUOUS-DOMAIN SIMULATION

As mentioned before, nowadays, more and more parallel programs run on workstation clusters. But there is still problem of low transmission performance in terms of bandwidth and latency of the network that connects the workstations.

Thus, only algorithms that guarantee a minimum communication overhead may be able to provide good speed-up on workstation cluster. Automatically parallelized code generally cannot achieve good speed-up.

Therefore, new algorithms have to be devised. *Domain decomposition* methods split the problem in the physical domain. Each of the obtained domains may be analyzed by a separate process, enabling parallelization. The boundaries of the domain are synchronized by a serial master process.

Process of splitting the electrical circuit into several pieces in order to simulate one piece per processor is referred to as *partitioning*.

The main objective of partitioning for parallel simulation is to reduce the simulation runtime. For low simulation runtimes it is crucial to achieve a low number of signals that connect the partitions. The reason for this is time consuming communication caused by the connecting signals. Parallel simulation is synchronized by a master process, which calculates the connection network serially and thus increases the runtime. Electronic circuits have natural clustering. This clustering can be used to achieve a good partitioning of the circuit.

Apart from low number of interconnecting signals, partitioning should provide equal workload for each slave processor, which would enable optimal distribution of simulation effort. In order to estimate the workload, each element is assigned a weight according to simulation complexity.

One of the first published methods for partitioning on a transistor level is *node tearing* [10]. Among other approaches are clustering algorithms such as building *DC connected components* and *strongly connected components*, or

diagonal dominance Norton partitioning [11]. Some of these methods are combined with *Fiduccia-Mattheyses method* [12] and *hierarchical methods* [13]. Another approach simply splits the ASCII file containing the circuit description and improves this initial partitioning by shifting components [14].

One of the modern partitioning methods is COPART [15], implemented in TITAN [16] parallel transistor level simulator. Other parallel circuit simulator Xyce [17] has number of partitioning algorithms implemented. One is ParMETIS [18], an MPI-based parallel library that implements a variety of algorithms for partitioning unstructured graphs, meshes, and for computing fill-reducing orderings of sparse matrices.

Once the circuit is partitioned, a parallel simulation should be performed.

Techniques for allowing each subcircuit to determine its own time steps and hence to optimize the simulation for each partition have been proposed [19], but most of the simulators use same time step for all the partitions.

The domain decomposition technique is often used for the parallelization of parallel differential equation solvers and obtains high performance. Set of nonlinear differential equations is discretized producing a set of nonlinear equations. Common parallel linear solver algorithm is *Schur Complement Technique*. It can be generalized using Newton's method to *Parallel Newton's Method* [16]. Using this algorithm each domain can be solved independently by performing the following steps in parallel without communication:

- model evaluation
- discretization
- calculation of the Jacobian matrix
- LU decomposition
- forward substitution
- calculation of outer derivative and right-hand side
- inner variables computation

Equation system representing the interconnections between the domains can't be solved in parallel, and are solved by master. As for the communication, coupling variables are sent to the slaves, and the local derivatives and the right-hand side are sent to the master.

This algorithm can be improved, as in *parallel multilevel Newton method with latency* [16], where nonlinearity is shifted to the slaves and iteration latency is exploited.

3. DISCRETE-EVENT SIMULATION

Exploiting parallelism in discrete-event simulations is particularly convenient because VHDL-like languages explicitly support descriptions consisting of concurrent processes.

In parallel discrete-event simulation (PDES), the system under simulation is modeled as a collection of concurrently executing logical processes (LPs) that communicate via message passing. LPs may be assigned to different processors, thus distributing the simulation across the network of workstations. Each message carries an event and a time stamp for the time when the corresponding event occurs in the simulated system. In order to perform the distributed simulation correctly each LP should process its input events in chronological order of their timestamps. There are two synchronization protocols used to ensure that: conservative and optimistic.

In conservative synchronization LPs process only "safe" events. Processes containing no safe events are blocked. An event is "safe" if it is impossible for the LP to receive another event with lower timestamp. Blocking may cause deadlock that can be avoided or detected and recovered by global synchronization.

In optimistic synchronization it is assumed that all the events are safe. Unlike conservative synchronization, an optimistically synchronized simulator is not restricted to simulate serially through the time. Different LPs can execute events at different simulation times simultaneously. In this type of synchronization each LP operates as a distinct discrete event simulator, maintaining input and output event lists, a state queue and a local simulation time. If an LP receives an event with a lower timestamp than its local simulation time (straggler), it must rollback to undo some work that has been done. During the rollback the LP restores the state previous to the straggler and cancels all the events sent during the wrong simulation by sending anti-messages. These anti-messages cause rollback at their destination LPs. Optimistically synchronized simulators can use either aggressive cancellation, in which all incorrect messages are discarded via anti-messages, or lazy cancellation, in which messages are only discarded when they are known to be incorrect. Lazy cancellation can improve simulation performance by decreasing the number of required rollbacks. However, if all messages are not canceled immediately, significant work may need to be discarded once the messages are eventually determined to be incorrect. After rollback, the events are re-executed in the correct chronological order. This requires that each process stores information about its previous states, inputs and outputs that can be used in case of a rollback. Therefore, optimistic protocols can cause memory overflow and global synchronization should be used to determine if a memory cell is old enough that can be freed. One such optimistic synchronization protocol is Time Warp used in implementation of mixed-signal VHDL-AMS simulator SEAMS [20]. In this simulator events are sent between LPs using MPI message passing standard.

Global synchronization between circuit partitions in the discrete-event simulation may be achieved using null-messages or global virtual time (GVT). A null message contains only timestamp without event and LPs use such messages to inform each other about their current simulation times. Global virtual time is the smallest timestamp of an unprocessed event in the whole system. It is monotonically increasing over the simulation. All history items with timestamps lower than GVT can be erased from memory.

Dynamic synchronization protocol that combines advantages of both conservative and optimistic synchronization methods, allowing processes to self-adapt for maximal utilization of concurrency is presented in [21, 22].

However, the PDES techniques applied so far has a number of drawbacks. This is due to the fact that each event that is processed will generate one or more events that must be communicated to other parallel processes resulting in high communication overhead. Some optimizations for improving the parallel logic simulation performance are given in [20]. These optimizations include circuit partitioning, roll-back relaxation and fine-grained communication optimizations. Circuit partitioning algorithms are used to divide the circuit to be simulated across LPs. The partitioning methods are based on

either parallelism in the simulation algorithm or in the circuit being simulated. Partitioning based on simulation algorithm is limited by the characteristics of specific algorithm used for simulation. The second method exploits the concurrency and parallelism in the circuit structure in order to minimize communication between LPs and balance the processor workloads. Many of the partitioning algorithms are based on a directed graph representation of the input circuit. In such representation the vertices of the circuit graph denote logic gates while edges represent signals. In ideal partitioning of a circuit graph LP workloads are ideally balanced and an equal number of gates are active at each simulation instance. Since each circuit has its own structure and pattern communication a specific partitioning algorithm cannot provide ideal partitioning. The multilevel approach to partitioning proposed in [23] optimizes all factors for improving parallel logic simulation by decoupling them into separate phases.

4. MIXED-MODE SIMULATION

A parallel simulation environment is very convenient for mixed-mode simulation, since a mixed-mode design is by default partitioned into analog and digital portions of the system.

In order to achieve distributed mixed-mode simulation a synchronization interface between analog and digital simulation kernels is required. Synchronization protocols supporting mixed-mode simulation in a distributed environment are presented in [24]. Discrete-event models are described using discrete-event processes whereas analog/continuous models are defined by differential equations. There are distinct points in simulation time where the communication between these models takes place. Appropriate interface functions are necessary to handle this communication and different notions of time should be addressed by the simulator. Discrete-event processes execute instantaneously as time is not advanced during execution. State changes occur at specific time points. Differential equation processes may advance the simulation time during execution. Such processes are called self-advancing processes. A mixed-mode simulator must enable an appropriate interaction between these different simulation processes. In [24] the process synchronization approach used to synchronize a Time Warp based parallel kernel with a continuous time differential equation simulation kernel is described. In this approach state saving is only required at synchronization points that decreases memory requirements. Two process based synchronization protocols are introduced: First Event Synchronization (FES) and Second Event Synchronization (SES). In the First Event Synchronization Protocol, the self-advancing process is handled as a discrete event process. That process is activated by the first event arriving at the start of the simulation interval (t_n). The process then executes to the time of the next scheduled event (t_{n+1}). If no event is generated, the self-advancing process calculates all intermediate values, stops execution at t_{n+1} and stores its state. At that moment, a new synchronization point is reached. In case an event is generated during computation of the internal values of the self-advancing process, the FES protocol requires state saving and an artificial event should be generated to force another synchronization point.

In the Second Event Synchronization protocol, synchronization is attempted on the receive time of the event determining the end of the self-advancing simulation interval. The self-advancing process is simulated to the time t_{n-1} representing the previous synchronization point. Then it is activated by the event at time t_n and continuous to simulate from t_{n-1} to t_n. If no new events are generated by self-advancing process the simulation stops at time t_n and new synchronization point is automatically generated. If an event is generated during the self-advancing simulation at time t'_n, the self-advancing process must be interrupted. The generated event at that time should be sent to discrete-event process and the discrete-event simulator should be notified that the self-advancing process did not complete the simulations up to time t_n. This problem can be solved by inserting a dummy event at time t'_n and make the system believe that the self-advancing process was activated by this event.

5. SIMULATOR PARALLELIZATION

In order to simulate complex mixed-signal electronic circuits at transistor level, they have to be modeled using algebraic equations and Ordinary Differential Equations (ODE).

At each iteration and at every time instant, the matrix entries of the system of linear equations have to be recalculated. These entries are derivatives of the nonlinear equations and are computed within separate subroutines. Having in mind the number of matrix entries (the system of 1000 variables has a matrix of up to 1 million entries – that may be reduced thanks to sparsity, but it is still very large), the number of iterations and the number of time instants, it is necessary to provide an immense computational effort. It has been shown in the literature that even for small systems, equation formulation takes more computational time than equation solution. Therefore, we propose to parallelize this task. In such kind of parallelization calculation of matrix contributions for nonlinear circuit elements is distributed to different cluster nodes and they are calculated simultaneously. At the same time master node calculates matrix entries for constant and linear time dependent elements. When generation of matrix entries for various nonlinear elements on different cluster nodes is finished, they are sent to the master node and the complete circuit matrix is formed. Then the master node should solve generated system of linear equations. That task can also be parallelized which leads to further reduction in simulation time.

Such parallelization requires neither a sophisticated task and circuit partitioning algorithm nor synchronization protocols, so it is easy to implement on a Beowulf cluster using MPI routines.

The presented parallelization of equation formulation process is implemented in the simulator Alecsis. It is a mixed-signal and mixed-domain simulator with proprietary hardware description language AleC++ capable for modeling and simulation of complex systems containing different kinds of devices and subsystems [25, 26]. The developed simulator with parallel simulation capability is called pAlecsis (*Parallel* Analog and Logic Electronic Simulation System).

The way of parallelization in the pAlecsis simulator is shown in Fig. 2. Parallel equation formulation during the simulation is implemented using one of the most common of parallel algorithm prototypes, master-slave algorithm [6]. The main idea is that one process, called the master process, is responsible for coordinating the work of the others (the slave processes). This mechanism is particularly suitable when the slave processes do not have to communicate with one another, which is the case in parallel matrix contribution calculations.

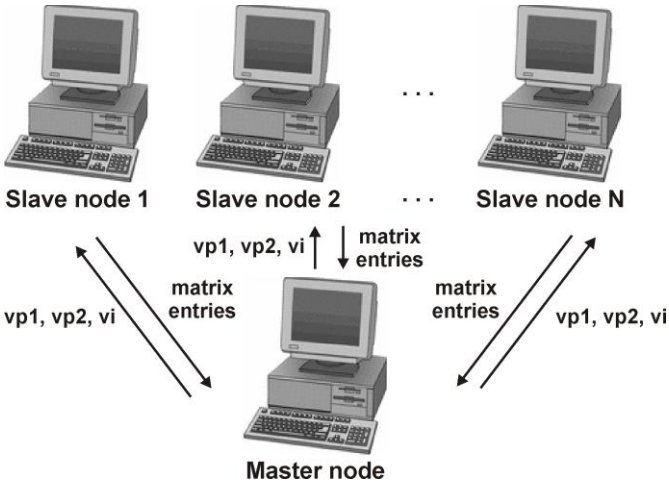

Fig. 2 Parallelization in the pAlecsis simulator on a Beowulf cluster

The calculation of matrix entries for nonlinear circuit elements (e.g. transistors) is distributed across master node and slave nodes of the cluster and performed in parallel. Since multiple cluster nodes calculate contributions for different elements simultaneously, the time necessary for equation formulation decreases. In order to minimize communication between cluster nodes, appropriate data structures for all elements of the circuit are generated on all nodes simultaneously during compilation of the AleC++ model. In that way all cluster nodes have the information necessary to generate matrix contributions for all elements. Each node of the cluster performs equation formulation and calculation of matrix entries for specific number of circuit elements. When entries for all elements on one slave are generated, they are sent to the master node using appropriate MPI routines (Fig. 2). The master node also calculates matrix entries for constant and linear time dependent elements. When the master node receives matrix entries from all slaves, it flushes them to the system matrix and performs one simulation step. In order to enable calculation of matrix entries on slave nodes, the master node should send vectors of solutions of the system of equations for the two past time instants and previous iteration. In Fig. 2 these vectors are denoted with *vp1*, *vp2* and *vi*. Appropriate MPI routines for transfering data are used to send and receive these vectors.

It is worth to mention here that the very solution of the system of simulataneous linear equations, that is generated in the way described above, is not parallelized yet. We intend to use SuperLU [27], [28] for this purpose in the first next step of improvement of pAlecsis.

Sequential simulation algorithms executing on a single workstation are tested for correctness usually by only seeing whether they give the right result. For parallel programs, that is not enough, but one wishes to reduce the simulation time. Therefore, measuring of simulation time is part of testing the parallel simulator to see whether it performs as intended. Usually performances of the parallel simulator are specifed as speedup. If parallel simulation executes on N single processor cluster nodes, speedup is normally defined as:

$$\text{Speedup} \quad \frac{\text{Simulation time on 1 node}}{\text{Simulation time on N nodes}} \qquad (1)$$

Implemented parallel simulation algorithm reduces simulation time for bigger circuits when time necessary to calculate matrix entries for all elements at every time instant and every iteration exceeds time necessary to calculate matrix entries on slave nodes *and send them to master node* over the interconnecting network. For such circuits the parallel simulation on the cluster is faster than the simulation on a single processor workstation.

In order to determine the size of circuits in number of transistors for which there is speedup in simulation on a cluster with two nodes, parallel simulations using the presented algorithm were performed on circuits consisting of various number of MOSFETs. These circuits are generated by successive replication of bilinear SC filter circuit with MOSFET operational amplifiers. Then speedup is calculated as simulation time on 1 node divided by simulation time on 2 nodes. The generated results are given in Table 1. When number of MOSFETs increases parallel simulation time increases slower than simulation time on 1 node, so parallel simulation time almost equals sequential simulation time for circuits containing 420 MOSFETs. For such and bigger circuits simulation time on 1 node is bigger than simulation time on 2 nodes, so the parallel simulator gives speedup in simulation (see Table 1).

Table 1: *Speedup of parallel simulation in pAlecsis*

Number of MOSFETs	Simulation Speedup Time(1 node) / Time(2 nodes)
420	0.96
740	**1.1**
1480	**1.5**

6. CONCLUSION

This paper presents the concept of gridification and parallelization of an electronic circuit mixed-mode simulator. Basic information regarding modern simulation, leading to the need for parallel simulation is presented. A survey of circuit partitioning techniques as well as equation solvers for parallel continuous domain simulation is given. Also, methods for logic circuit partinioning and parallel discrete-event synchronization protocols are considered. The problem of synchronizing parallel continuous-domain and discrete-event simulation krenels is addressed.

Implementation of a new algorithm that parallelizes equation formulation in pAlecsis simulator is presented. Simulation examples illustrate the speedup in simulation process using the presented algorithm. Further reduction in simulation time can be achieved by implementation of parallel solution of linear equations system. Such algorithms are well known in literature [27] and will be implemented in pAlecsis simulator.

In order to exploit advantages of grid computing, parallelization concepts implemented in pAlecsis simulator will be extended to gridification.

7. REFERENCE

[1] T. Sterling, "Beowulf Cluster Computing with Linux", MIT Press, 2001.

[2] I. Foster, C. Kesselmann, S. Tuecke, "The Anatomy of the Grid, Enabling Scalable Virtual Organizations", International J. Supercomputer Applications, 2001

[3] I. Foster, C. Kesselmann, "The Grid: Blueprint for a New Computing Infrastructure", 1998

[4] V. Litovski., and M. Zwolinski, "VLSI Circuit Simulation and Optimization", Chapman and Hall, London, 1997.

[5] Ž. Mrčarica, "Modelling of microelectromechanical devices and simulation of systems using hardware description language", PhD Thesis, TU Vienna, 1995.

[6] W. Gropp, E. Lusk, and A. Skjellum, "Using MPI: Portable Parallel programming with the Message-Passing Interface", second edition, MIT Press, 1999.

[7] W. Gropp, E. Lusk, and R. Thakur, "Using MPI-2: Advanced Features of the Message-Passing Interface", MIT Press, 1999.

[8] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard", International Journal of Supercomputer Applications, 8(3/4): 165-414, 1994.

[9] Message Passing Interface Forum, "MPI-2: A Message-Passing Interface Standard", International Journal of Supercomputer Applications, 12(1-2): 1-299, 1998.

[10] A. Sangiovanni-Vincentelli, L.-K. Chen, and L. O. Chua, "An efficient heuristic cluster algorithm for tearing large-scale networks", IEEE Transactions on Circuits and Systems CAS, CAS-24(12): 709-717, Dec. 1977.

[11] P. Debefve, F. Odeh, and A. E. Ruehli, "Waveform techniques", in Circuit Analysis, Simulation and Design, Part 2, Advances in CAD for VLSI, Vol. 3. A. E. Ruehli, Ed. Amsterdam, The Netherlands: Elsevier Science Publishers B. V., 1985, pp. 41-127.

[12] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions", in Proc. ACM / IEEE Design Automation Conf. (DAC), Vol. 19, 1982, pp. 175-181.

[13] P. Cox, R. Burch, and B. Epler, "Circuit partitioning for parallel processing", in Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD), 1986, pp. 186-189.

[14] T. Kage, F. Kawafuji, and J. Niitsuma, "A circuit partitioning approach for parallel circuit simulation", IEICE Transactions on Fundamentals, E77-A(3): 461-466, 1994.

[15] N. Fröhlich, V. Glöckel, J. Fleischmann, "A New Partitioning Method for Parallel Simulation of VLSI Circuits on Transistor Level", Proceedings of Design, Automation and Test in Europe 2000, pp. 679-684, Paris.

[16] N. Fröhlich, B.M. Riess, U. Wever, Q. Zheng, "A New Approach for Parallel Simulation of VLSI-Circuits on a Transistor Level", IEEE Transactions on Circuits and Systems, Part I, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 601-613, Vol. 45, No. 6, June 1998

[17] http://www.cs.sandia.gov/xyce/

[18] http://www-users.cs.umn.edu/~karypis/metis/parmetis

[19] M. Zwolinski, "The System Design of a Hierarchical VLSI Circuit Simulator", PhD Thesis, University of Southampton, 1986.

[20] D.E. Martin, R. Radhakrishnan, D. Rao, M. Chetlur, K. Subramani, P. Wilsey, "Analysis and Simulation of Mixed-Technology VLSI Systems", Journal of parallel and distributed computing, vol. 62, No 3, pp. 468-493, 2002.

[21] D. Lungeanu, C.J.R. Shi, "Parallel and Distributed VHDL Simulation", Proc. of the conference on design, automation and test in Europe, pp. 658-662, 2000.

[22] D. Lungeanu, C.J.R. Shi, "Distributed Simulation of VLSI Systems via Lookahead-Free Self-Adaptive Optimistic and Conservative Synchronization", Proc. of the 1999 IEEE/ACM international conference on computer-aided design, pp. 500-504, 1999.

[23] S. Subramanian, D. Rao, P. Wilsey, "Study of a Multilevel Approach to Partitioning for Parallel Logic Simulation", 14th International Parallel and Distributed Processing Symposium, pp. 833-836, May 2000.

[24] P. Frey, R. Radhakrishnan, "Parallel Mixed-Technology Simulation", Proc. of the 14th workshop on parallel and distributed simulation PADS'00, pp. 7-14, May 2000.

[25] Ž. Mrčarica et al., "Alecsis 2.3, the simulator for circuits and systems. User's Manual", Laboratory for Electronic Design Automation, Faculty of Electronic Engineering, University of Niš, Yugoslavia, LEDA – 1/1998.

[26] Ž. Mrčarica, T. Ilić, D. Glozić, V. Litovski, and H. Detter, "Mechatronic Simulation Using Alecsis: Anatomy of the Simulator", Proc. of the Eurosim '95, Vienna, Austria, pp. 651-656, 1995.

[27] X.S. Li, and J.W. Demmel, "A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems", ACM Trans. Mathematical Software, vol. 29, no. 2, pp. 110-140, June 2003.

[28] SuperLU, http://crd.lbl.gov/~xiaoye/SuperLU/

Садржај – *У овом раду представљен је концепт гридификације и паралелизације симулатора електронских кола са мешовитим сигналима. У раду су дате основне информације о модерној симулацији које доводе до потребе за паралелном симулацијом. Дат је преглед алгоритама за паралелну симулацију и постојеће имплементације. Описана је и имплементација новог алгоритма за паралелну формулацију једначина у симулатору pAlecsis.*

ГРИДИФИКАЦИЈА И ПАРАЛЕЛИЗАЦИЈА СИМУЛАТОРА ЕЛЕКТРОНСКИХ КОЛА

Марко Димитријевић, Бојан Анђелковић, Милан Савић, Ванчо Литовски