# Grid-Enabled Parallel Simulation Based on Parallel Equation Formulation

Bojan Andjelković, Vančo B. Litovski, and Volker Zerbe

**Parallel simulation is an efficient way to cope with long runtimes and high computational requirements in simulations of modern complex integrated electronic circuits and systems. This paper presents an algorithm for parallel simulation based on parallelization in equation formulation and simultaneous calculation of matrix contributions for nonlinear analog elements. In addition, the paper describes the development of a grid interface for a parallel simulator that enables a designer to perform simulations on distant computer clusters. Performances of the developed parallel simulation algorithm are evaluated by simulation of a microelectromechanical system.**

**Keywords: Parallel simulation, electronic circuit simulation, grid computing, Beowulf cluster, distributed computing, microelectromechanical systems.**

## I. Introduction

Rapid growth of the complexity in modern mixed-signal integrated circuit (IC) and system-on-a-chip (SoC) designs demands powerful simulators capable of quickly analyzing and validating the system's functionality as a whole. The simulation process of large-scale models describing today's ICs is memory and computationally intensive, and algorithmically complex. These properties pertain to the fact that a large number of ordinary (and, potentially, partial) nonlinear differential equations have to be solved for long running excitations. One should not forget that within some design activities such as test vector verification or circuit optimization, many of the repetitive simulations are expected to be performed for the same system. In addition, even a single simulation run creates a huge amount of data that needs to be processed. Because of all these reasons, simulation runtimes are very long and lead to slow design process.

An effective solution to this problem is to parallelize the simulation algorithm and use a distributed computing platform such as a cluster of workstations to run simulations. In such an approach, the simulator distributes complex calculations, necessary for the simulation process, across different workstations/processors, and executes them simultaneously.

The development of low-cost personal computers and gigabit LAN network connections offers a possibility for implementation of inexpensive distributed multiprocessor systems such as computer clusters. A cluster has many advantages over a classic supercomputer: it is inexpensive, flexible, easy to use, easy to maintain, and highly stackable. One particular implementation of this approach, involving open source system software and dedicated networks, has acquired the name "Beowulf" [1].

Manuscript received Mar. 31, 2009; revised last May 17, 2010; accepted June 1, 2010.

Bojan Andjelković (phone: +381 18 529321, email: abojan@gmail.com) was with the Faculty of Electronic Engineering, University of Niš, Serbia, and is now with Fujitsu Semiconductor Europe, Germany.

Vančo B. Litovski (email: vanco.litovski@elfak.ni.ac.rs) is with the Faculty of Electronic Engineering, University of Niš, Serbia.

Volker Zerbe (email: volker.zerbe@tu-ilmenau.de) is with the Ilmenau University of Technology, Germany.

doi:10.4218/etrij.10.0109.0197

  

The growth of Internet and WAN links of great capacity and speed led to development of the computational grid. In the same way as power grid provides electrical power, computational power can be obtained on demand from a network of providers, potentially belonging to the entire Internet. The grid is a highly heterogeneous and a geographically widely distributed computing system consisting of interconnected shared computer resources (computer clusters) that users can utilize for their demanding tasks. At the beginning, this paradigm had been strictly scientific and academic; but, as in the case of the Internet, it became widely accepted and popular. One of the most common definitions is that a computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities which enables on-demand access to computing, data, and services [2]. Grid computing is suitable for intensive calculations that require significant processing power, large operating memory, as well as storage capacity. The simulation of ICs is paradigmatic example of such calculations [3].

## 1. Related Work

Several parallel simulators of electronic circuits have been developed recently, such as Xyce [4], TITAN [5], and SEAMS [6]. FineSim Spice from Magma Inc. is a commercial transistor-level circuit simulator for mixed-signal SoCs that can run over distributed networks or multi-CPU workstations [7]. Several electronic design automation (EDA) companies have recently created parallel simulators for analog and mixed-signal designs running on multicore computer architectures. The simulators such as HSPICE [8] and Virtuoso Accelerated Parallel Simulator [9] use multithreading simulation capabilities to exploit multicore processors in simulation of large post-layout designs.

In order to minimize the simulation runtime, parallel simulators partition the data and computation over multiple workstations and processors/cores. The main goal of the applied partitioning algorithms is to minimize communication between the workstations/processors and achieve their equal load [10].

One of the first published algorithms for partitioning at transistor level is *Node Tearing* [11]. However, it was applied on a single-processor computer, and the main reason for its implementation was the intractable complexity of the whole system. This algorithm starts from an input voltage source and gathers adjacent elements until it reaches a specified partition size. If there are several possibilities for the selection of an adjacent element, the algorithm takes the node with fewer connections to other partitions.

Another approach for partitioning simply splits the ASCII file containing the circuit description and improves this initial partitioning by shifting components [12].

TITAN and Xyce are parallel transistor level simulators that use SPICE [13] as modeling language. TITAN uses a complex partitioning algorithm COPART to split the circuit description and distribute generated partitions to different workstations/processors. The partitioning is based on evaluating the level of coupling between adjacent elements, and the generated partitions (SPICE subcircuits) tend to have small number of interconnect nodes and well-balanced sizes [14]. Partitioning cuts the nets between partitions, that is, interconnect nets. Parallel analog simulator handles these nets as I/O nets for the related subcircuits and connects virtual voltage sources to these I/O's.

After partitioning of the circuit all parts are simulated in parallel as separate circuits. The partitioning algorithm also creates a master partition that takes care of the consistency of the circuit variables at the connections and accommodates local time agents to the main simulation time agent for all subcircuits. This "synchronization" is not to be confused with inter-processor synchronization that will be necessary in parallel simulation. The master process evaluates the network variables within the connection network while the slave processors wait (that is, stop the parallel simulation). The connection network calculation is time consuming, and simulation runtime increases with each additional interconnecting signal between the partitions [14].

In Xyce a single, a large nonlinear problem is distributed across a large number of processors on the matrix level. The partitioning is performed on the graph of the problem, after the problem has been flattened [15].

The amount of parallelism inherent to the circuit limits the partitioning success [6] and some of the algorithms give good results only on mutually independent subcircuits. However, in typical ICs, the elements are highly connected including strong feed-back signals, and partitioning generates many connections between the partitions causing significant inter-partition load and consequently longer simulation runtimes.

Techniques for allowing each subcircuit to determine its own time steps, and hence to optimize the simulation for each partition, have been proposed, too. However, most of the simulators use the same time step for all partitions due to the fact that an appropriate complex subcircuits synchronization algorithm is necessary to exchange required simulation data (values of voltage for interconnect nets) between the circuit partitions.

SEAMS is a VHDL-AMS simulator that implements parallel digital simulation. In order to perform parallel mixed-signal simulation, the parallel digital simulator is synchronized with an analog simulation kernel. Parallel analog simulation in SEAMS is based on grouping and solving for the unknowns

occurring in a connected set of equations [16]. A single connected set of equations is referred to as an "analog island." Partitioning is performed such that no two "analog islands" may communicate during simulation. However, it is very difficult, if not impossible, to determine sets of equations independent from each other, and the results of this algorithm may vary for different circuits and systems (for example, systems with strong feedback) depending on the properties of the equations describing them.

While a broad survey of various parallel simulator implementations and algorithms can be found in [17], in this paper, we present a new and original concept of development, a parallel simulation algorithm that could be implemented in a simulator to speedup the simulation process. This algorithm avoids all complex circuit partitioning algorithms which lead to simple parallelization. It is based on the fact that in any implementation of a circuit analysis algorithm, two steps are to be performed in a sequence: equation formulation and equation solution. Accordingly, parallelization is to be implemented in these steps separately. To come to the final idea on the concept of the new algorithm, we first claim that the problem of parallel solution of a system of equations is already solved in any respect. That means existing subroutines (IP software packages) for parallel solution of sparse systems of linear equations may be used and inserted into any simulation program. Second, being aware that all electronic circuits are nonlinear ones, one should have in mind that for these kind of circuits, the equation formulation step of the simulation algorithm requires much computational effort and time. Therefore, our algorithm implements parallelization of the equation formulation by distributing to different processors the calculation of contributions of the nonlinear elements to the system of equations. That leads to reduction in the overall simulation runtime. In order to explore the efficiency of the algorithm, it was implemented in a simulator running on a Beowulf type workstation cluster. A microelectromechanical system with a capacitive pressure sensor was used to evaluate performances of the developed parallel simulation algorithm. In addition, we describe the implementation of the grid interface for a parallel simulator that enables use of distant computer clusters on the grid network to perform simulations.

## II. Parallelization of the Simulation Algorithm

In order to simulate a complex mixed-signal and mixed-mode electronic system, it is necessary to describe (model) it. Mixed-mode systems are described using:

1. Algebraic equations
2. Ordinary differential equations (ODE)
3. Partial differential equations (PDE)
4. Abstract system-level behavioral descriptions
5. Logic functions (systems of discrete equations).

Algebraic equations describe the resistive, time independent part, whereas nonlinear ODEs describe the dynamic part of the system. Nonlinear ODEs are discretized using finite difference formulae (usually referred to as numerical integration rules) that reduce them to a system of nonlinear algebraic equations. In the simulation of microelectromechanical systems (MEMS), electronic systems containing transmission lines, and magnetomechanical components, or when electrothermal simulations are performed [18], PDEs are used for modeling these specific parts and phenomena within the system. In order to obtain a set of ordinary differential equations, space discretization is performed, introducing new time dependent variables distributed in space. Therefore, PDEs introduce new sets of ODEs, whose number is usually very large, depending on the discretization mesh [19]. The resulting ODEs are then discretized to produce a nonlinear algebraic equation system. All this generates a potentially large system of nonlinear equations to be solved at a large number of time instants depending on the properties of the system under simulation and the stimulus signals (number of events at the input). Thus, one comes down to the problem of formulation and solution of a system of nonlinear equations that is solved iteratively, with the help of linearization, that is, by application of Newton methods. There are two loops during the simulation flow: a time loop for incrementing simulation time, and inside it, an iterative loop for solving the system of nonlinear equations until convergence is reached. Since all electronic components (transistors, diodes etc.) are described by nonlinear models, the calculation of all derivatives that are necessary for the linearization (within the equation formulation phase) is a very time consuming part of the simulation process. The number of iterations per time instant depends on the properties of the system to be simulated, on the effectiveness of the so called predictor algorithm that creates initial solutions for the iterative process at every time instant, and on the equation formulation/solution algorithm that is implemented at the linearized level. Even for small systems, given a good initial solution, there are, in general, several iterations per time instant.

In circuit and system simulation, it is necessary to formulate the equations describing the system automatically. The modified nodal analysis (MNA) method is practically a standard for that purpose [20]. This method determines the contribution of every particular circuit element to the system of equations and enables automatic creation of the equations by successive addition of particular contributions in the circuit matrix and right-hand side vector. This means that in order to

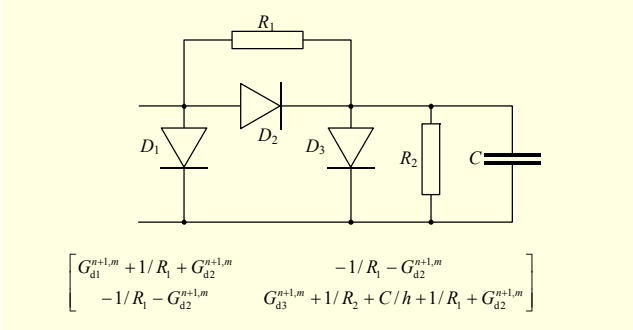


Fig. 1. Simple circuit with nonlinear elements and its matrix to illustrate equation formulation.

create system of linear equations in each iteration, one need to scan all elements in the circuit description, generate discretized models, generate linearized models for nonlinear elements, and finally add the contribution of every element to the matrix of the system of linear equations and right-hand side vector. In order to illustrate this, Fig. 1 shows a simple circuit containing three diodes. This figure also shows the circuit matrix with contribution of all elements.

Here $n$ denotes time steps counter (the time is calculated as $t=n\times h$ where $h$ is the time increment). $m$ denotes the number of iteration for a specific time instant. The simulator recalculates in each iteration contribution of diodes $G_{\mathrm{d}}^{n+1,m}=\partial i_{\mathrm{d}}/\partial u_{\mathrm{d}}=\lambda I_s e^{\lambda u_{\mathrm{d}}^{n+1,m}}$, which are derivatives of diode current per voltage between its terminals.

In the following paragraphs, we will consider and compare in more detail the existing algorithm with circuit partitioning and the new algorithm with parallel equation formulation. Figure 2(a) shows a typical algorithm for parallel simulation of nonlinear dynamic electronic circuits in a time domain. It is based on partitioning of the circuit model and simulation of the generated partitions in parallel [5]. As Fig. 2(a) shows, when the iterative analysis of a partition for a specific time instant finishes, the obtained result is not final. It is necessary for voltages at the connections between the partitions to be equal for each partition. In order to adjust these node voltages, one additional iterative loop is executed (denoted in the Fig. 2(a) as interconnect voltage adjustment loop). Therefore, there is an additional overhead in simulation time in this algorithm required to adjust interconnect voltages between the partitions.

Figure 2(b) shows a new approach to parallel simulation. In this algorithm no partitions are made and, consequently, it is not necessary to calculate interconnect voltages. To explain the idea we provide the following considerations.

During the simulation of a nonlinear dynamic electronic circuit at each iteration and time instant, it is necessary to recalculate the matrix entries of the system of linear equations. The entries coming from nonlinear elements are derivatives of

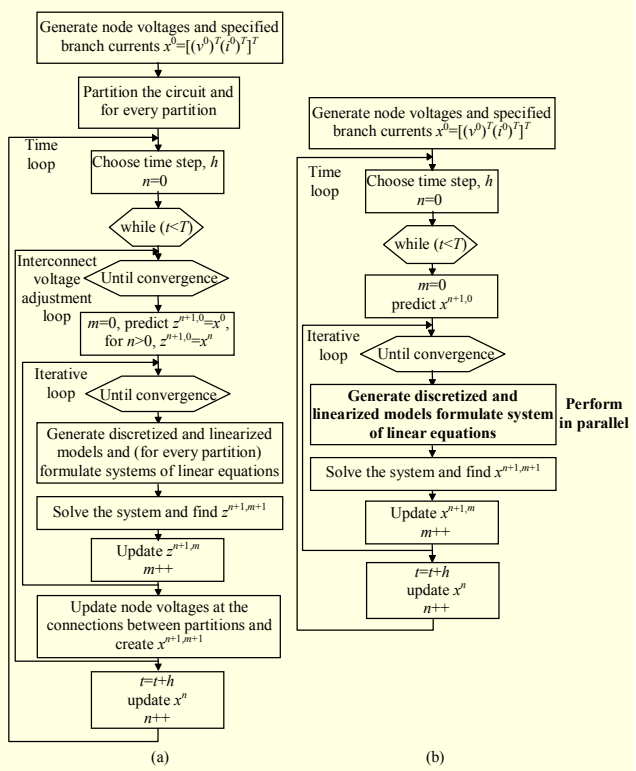


Fig. 2. Simulation algorithms for nonlinear dynamic circuits in time domain: (a) with partitioning of the circuit and (b) new algorithm based on parallel equation formulation.

the nonlinear equations, and the simulator computes them within specific subroutines.

A large number of matrix entries, iterations and time instants requires an immense computational effort during simulation. As demonstrated in [21], for small circuits with the number of nodes $N<20$, the majority of simulation time is spent during the equation formulation phase and forming the circuit matrix. When the size of the circuit grows, the time required for equation formulation increases linearly with the number of circuit elements, and therefore, with the number of equations used to model them. Therefore, the calculation of matrix entries and equation formulation for nonlinear circuit elements should be parallelized. The part of the simulation algorithm that it is possible to parallelize is highlighted in Fig. 2(b). This parallelization is an additional mechanism that can further accelerate the simulation together with parallelization of the solution phase of the system of linear equations that succeeds the equation formulation phase.

If one considers the circuit matrix as a sum of several matrices, the number of which is equal to the number of processors used in the simulation, it is possible to create the whole circuit matrix by creating its parts and then summing them. Figure 3 illustrates that process. Every submatrix contains entries for a specific (assigned) number of nonlinear

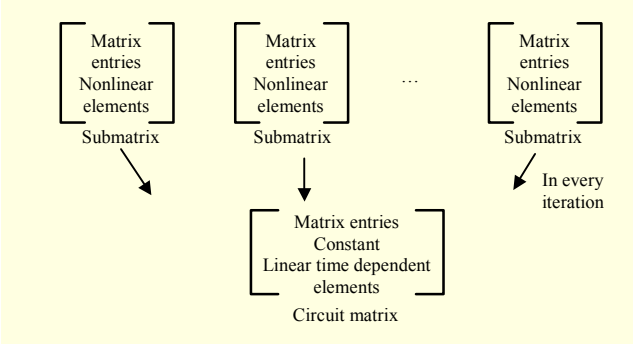


Fig. 3. Parallelization of equation formulation for nonlinear circuit elements.

circuit elements. There is no need for a special criterion for allocation of the circuit elements to a specific processor (that is a submatrix). The total list of nonlinear elements is simply divided into sublists with an equal number of nonlinear elements. The number of elements in a sublist is determined by simply dividing the total number of nonlinear circuit elements by the number of processors. Then, every processor calculates submatrix entries for elements in one partition.

Such parallelization of the simulation algorithm is a different approach from already developed solutions. The main differences between the new algorithm based on parallel equation formulation and the existing algorithms with circuit partitioning are:

- The new algorithm does not require sophisticated circuit and task partitioning algorithms to create partitions of equal size and minimal number of interconnect points. There is no partitioning of the circuit. However, the total number of nonlinear elements is divided into equal subsets which are assigned to processors.
- Since there is no partitioning, parallel simulation speedup does not depend on the circuit structure and number of interconnections between the partitions. Calculations of matrix contributions are completely independent of each other.
- There is no additional overhead in simulation time caused by the adjustment of interconnect voltages between partitions.
- There is no need for synchronization protocols to exchange interconnect data between the partitions, so the new algorithm is easy to implement on a distributed computing platform (for example, the Beowulf cluster).

The generation of matrix entries for linear resistive and linear dynamic elements is performed on one processor, since these calculations may be performed outside of the iterative loop. Moreover, the matrix contributions for linear resistive elements are calculated only once outside time and iterative loops, while entries for linear time dependent elements are calculated at every time instant outside iterative loop. All this reduces the overall time necessary for equation formulation. When parallel generation of matrix entries for all nonlinear elements is finished, the complete circuit matrix is formed (aggregated) as depicted in Fig. 3, and one may proceed with the solution of the system of linear equations (preferably in parallel).

It is necessary to note that this algorithm could be implemented in any simulator regardless of the hardware description language it uses. It introduces parallelization and speedup in the equation formulation phase that is independent of the language (Spice, VHDL-AMS, and Verilog-AMS) and type of models a designer uses to describe the system.

## III. Parallel Simulation Algorithm Implementation

The described parallelization of the equation formulation process is implemented in the simulator Alecsis [22]. It is a mixed-signal and mixed-domain simulator with proprietary hardware description language AleC++ [23] capable for modeling and simulation of complex systems containing different kinds of devices and subsystems [24]. Alecsis can handle various quantities appearing as physical connections between devices, from analog nodes and logic discrete signals, to non-electrical quantities such as pressure and light. AleC++ is an object-oriented hardware description language developed as a superset of C++. It provides some useful features, both for modeling hardware components and system-level descriptions, not found in other design languages [23]. Basic SPICE models are built-in (C-programmed) into Alecsis, while there is also a possibility to define semiconductor device models in AleC++. There are built-in models for diode, MOS transistor (models level 1, 2, 3, and BSIM model), bipolar transistor, and JFET. In addition, there are built-in primitive components such as resistor, capacitor, inductance, and voltage and current generators. AleC++ allows usage of SPICE model cards with the same syntax as in SPICE.

One possibility to model analog (continuous) devices in AleC++ is using equation statements in a similar manner as in VHDL-AMS. This gives a designer absolute freedom in modeling since the contribution of the model to the system of equations is directly given. Equations consist of regular expressions and may contain operators for first- and second-order time derivatives. Alecsis uses Gear methods for numerical integration of ordinary differential equations, Newton-Raphson method for nonlinear equations, and modified Berry's algorithm for solving system of linear equations characterized by sparse matrices [20].

Figure 4 shows the implementation of the described parallel simulation algorithm in the Alecsis simulator on a Beowulf

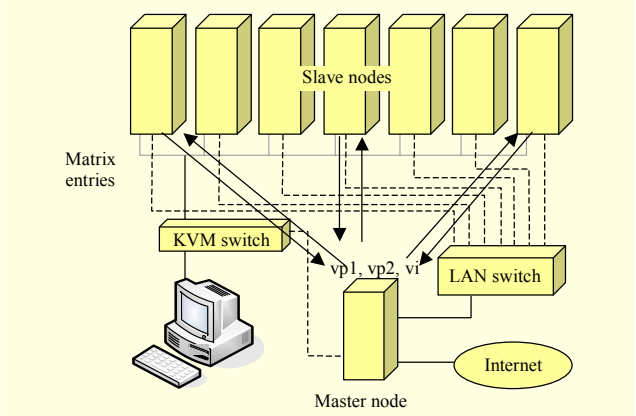


Fig. 4. Implementation of the parallel simulation algorithm on a Beowulf cluster.

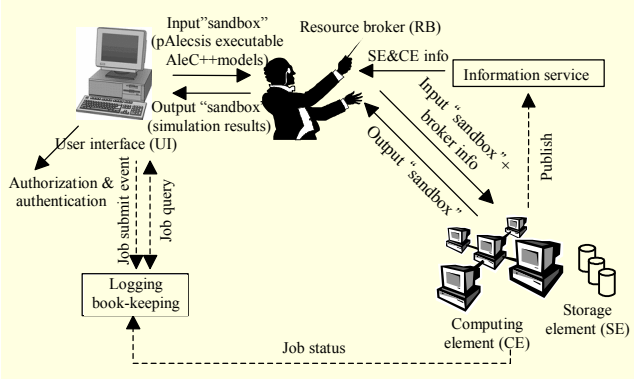


Fig. 5. Using grid resources for parallel simulations.

cluster using MPI routines [25], [26]. The simulator with added parallel simulation capability is called the parallel analog and logic electronic circuits simulation system (pAlecsis).

The presented parallel equation formulation is implemented using the master-slave algorithm [25]. In this algorithm, different cluster nodes (slave nodes) simultaneously calculate matrix entries for nonlinear circuit elements per time and per iteration. At the same time, the master node calculates matrix entries for specific number of nonlinear elements, as well as for linear resistive and linear time dependent elements.

To achieve an equal load for all nodes, each node of the cluster performs equation formulation and calculation of matrix entries for equal number of nonlinear circuit elements and stores the generated matrix entries in a linked list of data structures. These structures contain the value of the matrix entry, as well as information about matrix column and row to which the value contributes. After generation of entries (for all elements) on one slave, they are packed in an array and sent to the master node using message send and receive functions compliant with the MPI message passing standard (Fig. 4). Such aggregation of communication messages in larger groups before sending them across the network minimizes communication overhead [6].

When the master node receives matrix entries from all the slaves, it flushes them to the circuit matrix and performs one iterative simulation step. In order to enable calculation of matrix entries on slave nodes, the master node should send the slaves vectors of solutions of the system of equations for the two past time instants and previous iteration (denoted by vp1, vp2, and vi, respectively in Fig. 4). Appropriate MPI routines for transferring data are used to send and receive these vectors.

## IV. Enabling Parallel Simulations on the Grid

The computational grid attempts to provide access to powerful computing resources using wide area connections. The hardware part of the grid infrastructure consists of a number of computer clusters containing various numbers 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 task submission and management, file transfer, database access, data management, and monitoring. They also ensure security in multiuser environment using certificates.

To enable a designer to run parallel simulations using the grid resources, it is necessary to develop an appropriate grid interface for a simulator. Such an interface should provide submission of simulation jobs together with simulation models, simulation run on a distant computer cluster and retrieval of simulation results.

Figure 5 shows the implementation of the grid interface for the pAlecsis simulator that enables it to perform parallel simulations using the grid infrastructure. In order to execute simulations using grid resources, the user should submit a simulation job to a resource broker (RB). Simulation jobs are described in the job description language (JDL) [27]. The sample JDL file for simulations in pAlecsis on the grid has the following structure:

```
#Type of JDL
Type = "Job";
#Type of Job
JobType = "MPICH";
#Number of Nodes
NodeNumber = 3;
# The command/executable to run on the CE
Executable = "MPI_alec.sh"
# Arguments that are needed by the executable
Arguments = "circuit.ac";
# Files to be transferred before the job execution
InputSandbox = {"MPI_alec.sh","circuit.ac"};
#The files to be transfered after the job execution
OutputSandbox = {"mpiexec.out","circuit.ar"}.
```

The job description specifies the file(s) containing the model for simulation described in AleC++ (circuit.a*c*). This set of files is called the input sandbox, and it is initially copied to an RB.

The job description also specifies a minimal number of cluster nodes for simulation. RB looks for the best available computing element (CE) to execute the job. Since pAlecsis needs MPI support on a Beowulf cluster to run simulations, it is necessary to specify in the job description file that simulations are an MPI job type ("MPICH" job type). It informs the RB to look for CEs with installed MPICH runtime environment. If the simulation ends without errors, the output files containing the results of the simulation and specified by the user (circuit.ar) in the so-called output sandbox in the JDL file are transferred back to the RB node. At this point, the user can retrieve the output of his simulation. There is also a file containing information about the names and number of cluster nodes on the grid where the simulation executed (mpiexec.out).

The developed grid interface enables use of a cluster on the grid to perform complex, resource demanding simulations for which a designer does not have appropriate computer equipment locally. Since the simulation executes on a grid cluster in the same way as on the local workstation cluster, the only overhead in using the grid is additional time necessary to submit the simulation job and retrieve simulation results from RB.

## V. Parallel Simulation Algorithm Performance Evaluation

Performances of the presented parallel simulation algorithm can be evaluated using simulation speedup. If a parallel simulation executes on $N$ single-processor cluster nodes, simulation speedup is defined as

$$\text{Speedup} = \frac{\text{Simulation time on single processor}}{\text{Simulation time on } N \text{ nodes}}. \quad (1)$$

Performances of the parallel simulation using parallel equation formulation for electronic circuits of various sizes are explored in [28].

Figure 6 shows a capacitive pressure sensor with switched-capacitor (SC) circuit for converting the capacitance value into the output voltage [29]. This system can be used as a benchmark to evaluate performances of the proposed parallel simulation algorithm. $\phi_1$ and $\phi_2$ are the switching (digital) controlling signals. The capacitive pressure sensor is a rectangular membrane with all four edges built into the rim. If pressure is applied, the capacitance between the membrane and the bottom plane changes.

The behavior of the pressure sensor is described by the following equation [19]:

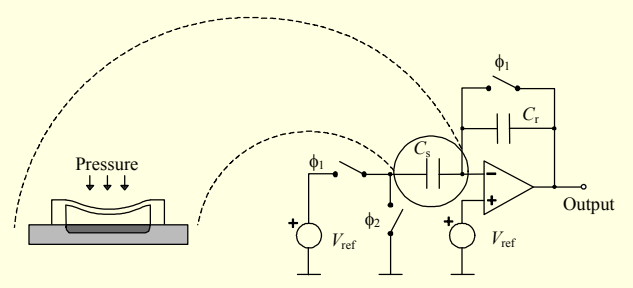


Fig. 6. Capacitive pressure sensor with SC read-out electronics.

$$D\frac{\partial^4 w}{\partial x^4} + 2D\frac{\partial^4 w}{\partial x^2 \partial y^2} + D\frac{\partial^4 w}{\partial y^4} = p + k\frac{dw}{dt} + ph\frac{d^2 w}{dt^2}, \quad (2)$$

where $w$ is the displacement of the membrane, $x$ and $y$ are the spatial variables, $D$ is the bending rigidity, $p$ is the pressure, $k$ is the damping coefficient, $\rho$ is the material density, and $h$ is the membrane thickness.

If the sensor has a rectangular shape with dimensions $L$ in $x$ direction and $W$ in $y$ direction, then the boundary conditions for built-in edges are:

$$\begin{aligned} (w)_{x=0 \vee x=L} &= 0, \quad \left(\frac{\partial w}{\partial x}\right)_{x=0 \vee x=L} = 0, \\ (w)_{y=0 \vee y=W} &= 0, \quad \left(\frac{\partial w}{\partial y}\right)_{y=0 \vee y=W} = 0. \end{aligned} \quad (3)$$

Therefore, the problem of membrane simulation is solving (2) belonging to the category of PDEs. In order to do that it is necessary to discretize it by one of the existing methods. Here we use the simplest one referred to as finite difference method [30]. It replaces partial derivatives with finite differences of close points of the membrane [19]. The discretization is performed by defining a spatial mesh, and one equation is defined for every mesh point. The same procedure is applied to all PDEs representing boundary conditions. Applied discretization formula uses 12 neighboring points for spatial discretization of (2) (shown as white circles in Fig. 7). Thus, PDEs for the membrane and boundary conditions are reduced to a system of ODEs. The solution of (2) is the vector $w$ for all points of the mesh.

The capacitance of the pressure sensor is calculated as follows [19]:

$$C = \varepsilon \int_{x=0}^{L} \int_{y=0}^{W} \frac{dxdy}{l - w(x,y)}, \quad (4)$$

where $l$ is the distance between the electrodes when no pressure is applied. This equation is nonlinear and it is possible to discretize it using the previous set of mesh points for the membrane. Space discretization gives

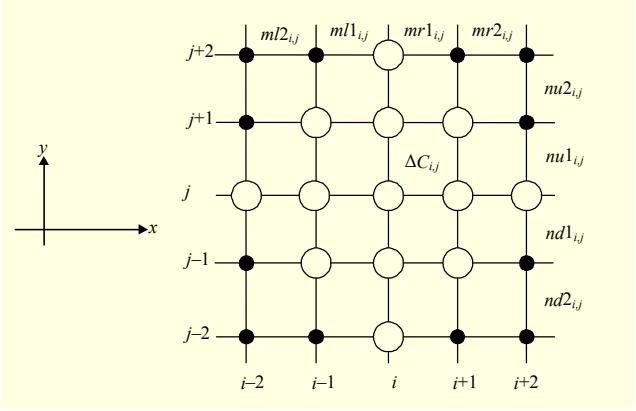


Fig. 7. Mesh points for spatial discretization of the membrane.

$$\Delta C_{i,j} = \varepsilon \frac{m_{i,j} \cdot n_{i,j}}{l - w_{i,j}},$$
$$m_{i,j} = \frac{ml1_{i,j} + mr1_{i,j}}{2}, \quad n_{i,j} = \frac{nu1_{i,j} + nd1_{i,j}}{2}, \tag{5}$$

where $\Delta C_{ij}$ is a part of the total capacitance corresponding to one mesh point. $ml1_{ij}$ and $mr1_{ij}$ are discretization steps in $x$ direction, while $nu1_{ij}$ and $nd1_{ij}$ are discretization steps in $y$ direction for mesh point $(i, j)$. $\varepsilon$ is dielectric constant of the gas in the sensor chamber. The equation for $\Delta C_{ij}$ is repeated for every mesh point $(i, j)$ (Fig. 7), and all these equations contribute to the same equation representing the total capacitance of the sensor:

$$C = \sum_i \sum_j \Delta C_{i,j}. \tag{6}$$

The simulation algorithm of the system with capacitive pressure sensor in Fig. 6 contains one more loop than the algorithm in Fig. 2(b) relating to spatial discretization. It generates a large number of nonlinear dynamic elements described by (5) for every mesh point that contribute to the total capacitance of the sensor. In order to simulate the membrane behavior more precisely, it is necessary to use small discretization steps in the mesh. It leads to a huge number of nonlinear equations for the sensor capacitance. Therefore, application of the previously described parallelization of equation formulation for nonlinear elements is very appropriate. Slave nodes of the cluster simultaneously generate contributions (entries in the system matrix) for an equal number of nonlinear elements given by (5).

The equation describing the electrical behavior of the capacitive pressure sensor is

$$i = \frac{dQ}{dt} = \frac{dC}{dt} v + C \frac{dv}{dt}, \tag{7}$$

where $Q$ is the charge on the capacitor electrodes and $v$ is the voltage across its terminals. The equation formulation process for implementation of this model is described in detail in [20]. The complete model of the sensor contains two electrical terminals for connecting the sensor plates, and one mechanical terminal (pressure).

Table 1. Beowulf cluster structure.

| | |
|---|---|
| Master node | PC Pentium IV,<br>2.4 GHz, 1 GB RAM, 240 GB HDD |
| Slave nodes | 4 X PC Pentium IV,<br>2.4 GHz, 512 MB RAM, 80 GB HDD |
| Interconnecting LAN | 1 Gbit/s Ethernet |
| Operating system | Scientific Linux |

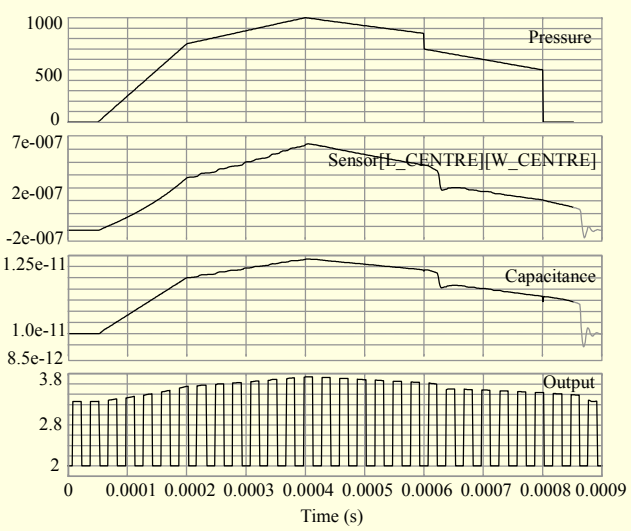


Fig. 8. Simulation results of the capacitive pressure sensor with SC read-out electronics depicted in Fig. 6.

Figure 8 shows simulation results of the system in Fig. 6. The traced quantities are the pressure, the displacement of the membrane center, the capacitance of the sensor, and the circuit output voltage.

Table 1 shows the structure of the Beowulf cluster used to perform parallel simulations.

There are 2,500 discretization points for the membrane of the sensor in the system in Fig. 6 used to evaluate parallel simulation performances. It generates 2,400 nonlinear elements for calculating the sensor capacitance.

Figure 9 shows parallel simulation speedup achieved for various numbers of cluster nodes involved in the simulation of this system. Since the simulation time is affected by overheads that are defined by factors outside of the simulation algorithm (time taken by the operating system, for example) the simulations were repeated five times, and the average was used to represent the speedup. The implemented parallel simulation

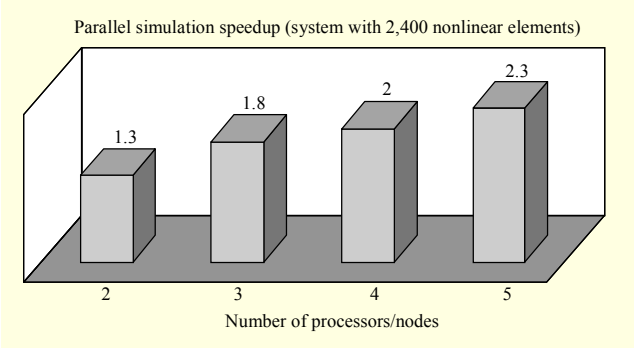


Fig. 9. Parallel simulation speedup of the system in Fig. 6 for various numbers of cluster nodes.

algorithm comes in effect (reduces simulation time, that is, increases simulation speedup) for a bigger number of discretization points (that is, a bigger number of nonlinear elements contributing to the total sensor capacitance). Actually, in this case the time necessary to calculate matrix entries for all nonlinear elements per time and per iteration exceeds time needed to calculate matrix entries on slave nodes and send them to the master node. It enables the parallel simulation on the cluster to outperform the sequential simulation on a single processor workstation. Since the master and slave nodes calculate matrix entries for an equal number of nonlinear elements, there is an equal load of all workstations during the phase of equation formulation. In the phase of equation solution, only the master node is active, since parallel solution of the system of linear equations is not implemented. If the simulator also solves the generated linear system in parallel, the load of workstations will depend on the applied algorithm. One should not forget that the speedup described in this paper is related only to parallelization of equation formulation, without applying a parallel solution of the system of linear equations.

Although the cluster of 5 CPUs is small, it could be used to show the parallel simulation speedup. As reported in [31], many legacy EDA applications show speedup only when parallelized on 4-8 CPUs. Parallelization of EDA applications cannot be compared to parallelization of the applications performing big number of independent complex calculations which can achieve higher speedup if calculations are simply distributed on a higher number of computers/CPUs. The algorithms used in EDA applications are very complex, and certain communication between CPUs cannot be avoided. This limits the speedup on a high number of computers/CPUs.

Comparison of these results to the simulation results of other parallel simulators requires performing the described simulation using different simulators on the same cluster. However, these simulators are proprietary software and not freely available. Also, some of them are not capable of simulating MEMS described by partial differential equations. Moreover, since the focus of this paper is the parallel simulation algorithm and not a specific simulator implementing such an algorithm, the paper does not discuss any comparison between the presented simulator and other parallel simulators.

## VI. Conclusion

This paper describes the concept of a parallel simulation algorithm for circuits and systems. It is easy to implement such an algorithm in a simulator running on a distributed computing platform. The algorithm introduces parallelization in the equation formulation phase for nonlinear analog elements. It enables distribution of calculations of matrix entries for nonlinear elements across different cluster nodes/processors. Therefore, the time needed for equation formulation decreases, reducing the overall simulation time. Such parallelization of the simulation algorithm is different from the solutions already developed because it does not require complex partitioning algorithms or a synchronization protocol between the partitions. In this way, parallel simulation speedup does not depend on the structure of the system under simulation and number of interconnecting signals between the partitions.

In addition, this paper describes the development of grid interface for a parallel simulator. It enables a designer to perform demanding simulation tasks not just on the local computer cluster, but also on distant shared clusters connected to the grid. The interface provides all necessary features to execute simulations in the grid environment, such as submission of simulations jobs and models, monitoring of simulation tasks, and retrieval of simulation results.

Performances of the developed parallel simulation algorithm are explored in simulation of a system with capacitive pressure sensor. The description of such a microelectromechanical system contains a large number of nonlinear analog elements because of spatial discretization of the sensor membrane.

The development of the pAlecsis simulator is a part of Southeastern European grid-enabled infrastructure development 2 (SEE-GRID-2) project co-funded by the European Commission under the FP6 research program.

## References

[1] T. Sterling, *Beowulf Cluster Computing with Linux*, MIT Press, Cambridge, Massachusetts, 2001.

[2] I. Foster and C. Kesselmann, *The Grid: Blueprint for a New Computing Infrastructure*, San Francisco, CA: Morgan Kaufmann, 1999.

[3] J.A.B. Fortes, R.J. Figueiredo, and M.S. Lundstrom, "Virtual Computing Infrastructures for Nanoelectronics Simulation," *Proc.*

*IEEE*, vol. 93, no. 10, Oct. 2005, pp. 1839-1847.

[4] Xyce Parallel Electronic Simulator home page, Available: http://www.cs.sandia.gov/xyce/

[5] N. Fröhlich et al., "A New Approach for Parallel Simulation of VLSI-Circuits on a Transistor Level," *IEEE Trans. Circuits Syst. I*, vol. 45, no. 6, 1998, pp. 601-613.

[6] D.E. Martin et al., "Analysis and Simulation of Mixed-Technology VLSI Systems," *J. Parallel Distributed Computing*, vol. 62, no. 3, 2002, pp. 468-493.

[7] "Magma Touts First Parallel Fast Spice," *EE Times*, Mar. 2007, Available: http://www.eetasia.com/ART_8800456053_480100_NP_c6fee15f.HTM

[8] "Enhanced HSPICE Revs up Circuit Simulation," *EE Times*, Mar. 2008, Available: http://www.eetasia.com/ART_8800510252_499495_NP_3ebd77e7.HTM

[9] "Virtuoso Accelerated Parallel Simulator" Available: http://www.cadence.com/products/cic/accelerated_parallel/pages/default.aspx

[10] M. Wolf and E. Boman, "Parallel Processing '08: An Increasing Role for Combinatorial Methods in Large-Scale Parallel Simulations," June 2008, Available: http://www.siam.org/news/news. php?id=1378.

[11] A. Sangiovanni-Vincentelli, C. Li-Kuan, and L. Chua, "An Efficient Heuristic Cluster Algorithm for Tearing Large-Scale Networks," *IEEE Trans. Circuits Syst.*, vol. 24, no. 12, Dec. 1977, pp. 709-717.

[12] T. Kage, F. Kawafuji, and J. Niitsuma, "A Circuit Partitioning Approach for Parallel Circuit Simulation," *IEICE Trans. Fundamentals E77-A(3)*, 1994, pp. 461-466.

[13] L.W. Nagel, *SPICE 2, a Computer Program to Simulate Semiconductor Circuits, Memorandum ERL-M250*, University of California, Berkley Press, 1975.

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

[15] Zoltan: Parallel Partitioning, Load Balancing and Data-Management Services, Available: http://www.cs.sandia.gov/Zoltan/

[16] P. Frey et al., "SEAMS: Simulation Environment for VHDL-AMS," *Proc. Winter Simulation Conf.*, 1998, pp. 539-546.

[17] M. Dimitrijević et al., "Gridification and Parallelization of Electronic Circuit Simulator," *Proc. INDEL, Conf. Ind. Electron.*, 2006, pp. 95-100.

[18] M. Jakovljević et al., "Transient Electro-Thermal Simulation of Microsystems with Space-Continuous Thermal Models in Analogue Behavioural Simulator," *Microelectronics and Reliability*, vol. 40, no. 3, 2000, pp. 507-516.

[19] Ž. Mrčarica, "Modelling of Microelectromechanical Devices and Simulation of Systems Using Hardware Description Language," (doctoral dissertation), Technical University Vienna, 1995.

[20] V. Litovski and M. Zwolinski, *VLSI Circuit Simulation Optimization*, London: Chapman and Hall, 1997.

[21] A.R. Newton and A.L. Sangiovanni-Vincentelli, "Relaxation-Based Electrical Simulation," *IEEE Trans. Electron. Devices*, vol. 30, no. 9, Sept. 1983, pp. 1184-1207.

[22] D. Glozić et al., "Alecsis, the Simulator," Laboratory for Electronic Design Automation, Faculty of Electronic Engineering, University of Niš, Serbia, 1996, Available: http://leda.elfak.ni.ac.yu/ projects/Alecsis/alecsis.htm

[23] V. Litovski, D. Maksimović, and Ž. Mrčarica, "Mixed-Signal Modeling with AleC++: Specific Features of the HDL," *Simulation Practice and Theory*, vol. 8, 2001, pp. 433-449.

[24] Ž. Mrčarica et al., "Mechatronic Simulation Using Alecsis: Anatomy of the Simulator," *Proc. Eurosim*, 1995, pp. 651-656.

[25] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel programming with the Message-Passing Interface,* 2nd ed., Cambridge, Massachusetts: MIT Press, 1999.

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

[27] S. Burke et al., "gLite3 User Guide," Available: http://edms.cern.ch/ file/722398/1.1/gLite-3-UserGuide.pdf, 2007

[28] B. Anđelković, V. Litovski, and P. Petković, "Implementation and Performance Analysis of Parallel Circuit Simulator on Beowulf Cluster," *Proc. ETRAN Conf.*, 2007, (*Proc. on CD, EL1.6.*)

[29] Ž. Mrčarica et al., "Describing Space-Continuous Models of Microelectromechanical Devices for Behavioural Simulation," *Proc. EURO-DAC Euro. Design Automation Conf. with EURO-VHDL*, 1996, pp. 316-321.

[30] Ž. Mrčarica et al., "Integrated Simulator for MEMS Using FEM Implementation in AHDL and Frontal Solver for Large-Sparse System of Equations," *Proc. Design Test Microsyst.*, 1999, pp. 271-278.

[31] "Intel TV Interview: Parallelizing Legacy EDA Applications," Available: http://www.cadence.com/Community/blogs/ii/archive/2009/12/16/intel-tv-interview-parallelizing-legacy-eda-applications.aspx

**Bojan Andjelković** received his MSc from the Faculty of Electronic Engineering, University of Niš, Serbia, in 1999. He worked there as a teaching and research assistant until 2007. His research interests include the parallelization of EDA applications and SoC design. Currently he works as a design engineer for Fujitsu Semiconductor Europe, Germany.

**Vančo B. Litovski** received his BSc, MSc, and PhD from the Faculty of Electronic Engineering, University of Niš, Serbia. His research interests include electronic filters, electronic circuits, systems modeling and simulation, and integrated circuits design. He is the author of several hundred scientific papers and scores of scholarly books. He received the Tesla and the Savastano Awards.

**Volker Zerbe** received his MS (Diploma) from the Technical University, Sofia (Bulgaria), in 1986. From 1986 to 1991, as research assistant, he developed system software for multiprocessor robot systems. In 1991, he received his PhD from the Ilmenau University of Technology, Germany. Since 1991, he has been a senior assistant at the Department of System and Software Engineering. Between 1992/1993 and 2007/2008, he was the Head of the Department at the Ilmenau University of Technology. His research interests include model-based mission/system design of embedded systems, design of parallel systems, and avionic and automotive applications.