A Static Partitioning Tool for Circuit Simulation

Luís Miguel Silveira

INESC/IST

R. Alves Redol, 9

1000 Lisboa

PORTUGAL

Abstract

This report sumarizes the main ideas and the strategy followed in the development of a static partitioning tool for a circuit simulator. This pre-processor tool will be used in an implementation of the circuit simulator CINNAMON on a multiprocessor machine.

This work was partially supported by the Esprit 1058 *Project*

Contents

1 Partitioning for Circuit Simulation

Partitioning is a technique related to the "divide-and-conquer" strategy. The basic idea is to break a large problem, into a set of smaller problems and to solve them one at a time.

In circuit simulation, partitioning has been widely used, in order to deal with the large amounts of data existing. To partition a circuit into several sub-circuits can be done either statically or dynamically. Static partitioning, based on circuit topology only, is faster but some times has problems dealing efficiently with the dynamics of the circuit itself. Dynamic partitioning, on the other hand, can solve this problem, provided that the overhead introduced doesn't become too expensive.

The strategy described in this report is a static one. It will be implemented in **CINNAMON** [1], and used in the first multiprocessor implementation of the simulator. Data read from the simulation input file, will be partitioned in a set of "sub-networks" that will be the higher data-level for the multiprocessor implementation.

CINNAMON's Parallel version 1 will be based on a **static data partitioning strategy**. The basic ideia for this partitioning is closely related to the event-driven nature of the algorithm and tries to explore a little further the concept of **localized activity** [2]. Changes in one area of the circuit cause, at first, only local disturbance, that will take some time to propagate to further areas. This concept leads to a natural way of solving the equations and to the use of event-driven techniques to control the process of simulation [3]. If multiple localized activity centers exist, they can be dealt with in parallel with little or no interaction.

If the result of data partitioning is a set of strongly connected “sub-networks” with a minimum of interaction, then internal changes in one “sub-network” can be dealt with locally and only later will appear on the “border” of the “sub-network”. Only then, they will have to be propagated to the “neighbor sub-networks”. This is the main ideia on which data-partitioning techniques are based.

In the next chapter the static partitioning tool developped for **CINNAMON** will be described. Later, we will relate it to the work to be developped, and to the parallel implementation of **CINNAMON** to be started.

2 A Static Partitioning Tool for Circuit Simulation

The partitioning strategy implemented in the pre-processor tool was based on the "disturbance locality" concept introduced [1]. Communication between sub-networks is therefore only needed when that disturbance has to be propagated to neighbor sub-networks.

An essential concept for this partitioning strategy emerges: the notion of **sub-network border**. The **border** is the set of nodes from a sub-network that have connections to foreign nodes, i.e., nodes in other sub-networks. A node belonging to a **sub-network border** will be called a **border** node.

In the algorithm currently implemented in **CINNAMON**, the solution of an event on a node, implies solving a system of equations for that node and its adjacents. Therefore, when solving an event on a node adjacent to a **border** node, we will need data from outside the sub-network. We will thus call **nearborder** nodes, to the nodes adjacents to the sub-network border and distinguish them from the other nodes in the sub-network that we will call **internal**.

Obviously voltage source nodes will not to be considered as belonging to any sub-network. Events on source nodes are different because they cause adjacent nodes to be calculated but are not themselves calculated (their changes are known "a priori"). Voltage source nodes have no need for a special tag but for the sake of uniformity we will call them **external** (although in a strict sense, they are also **border** nodes).

---

[1]We acknowledge the help of Paulo Flores in the development of the partitioning tool

We will thus consider in a circuit, four different kinds of nodes in a circuit, when considering data partitioning: **external**, **internal**, **border** and **nearborder**.

The higher level of the data hierarchy is therefore sub-network. We need, for each **border** node a list of connected sub-networks (rather than a list of connected nodes that is nontheless needed when solving events). This is also true for voltage source nodes.

The existence of that information associated with all border nodes will allow changes in a sub-network, that arise in its border, to be propagated to the rest of the circuit. An array of structures is created at partitioning time containing, for each node, the number of the sub-network it belongs to, the type of node and, if it is a **border** node, a list of neighbor sub-networks to be warned when that node changes.

The main issue that remains to be discussed is what a sub-network should be. Generaly speaking it should be a set of strongly connected nodes with few connections to the exterior.

A typical way of partitioning MOS circuits is to divide the network in sets of channel-connected nodes [4] (in the following, the terminology "groups" will be used to denote this partitioning strategy).

This strategy leads typically to very small "groups". In our **coarse-grain** parallel version of **CINNAMON** we are not interested in small "groups" because in that case each sub-network would have few internal nodes and most of the time would be wasted communicating with others groups.

In order to speedup the convergence of DC analysis in **CINNAMON**, a switch-level like analysis of the circuit is performed, based upon a static partitioning of the data read. Based on that partitioning algorithm, the code was partially rewritten so that extra information is kept for each “group”, like its size, its neighbor groups, etc. New routines were developed that will use this information to regroup the data into an arbitrary number of “sub-networks”. These “sub-networks” will be formed by the recollection of “groups” and so that the groups inside a “sub-network” will be connected. In this manner versatility and compatibility are kept.

The algorithm that gathers “groups” into “sub-networks” is a “greedy” algorithm that sees each “group” as a node in a graph and proceeds by gathering connected “groups”, keeping the balancing between the number of nodes in each “sub-network”, and returning a specified number of subgraphs, the “sub-networks”. Each “sub-network” can be viewed, internally, as a connected graph itself.

Ideally the gathering of “groups” into “sub-networks” should minimize the communication between “sub-networks”. The research for that type of algorithms is nevertheless beyond the scope of this work.

If the partitioning algorithm returns sub-networks of the size of groups consisting of only channel-connected nodes (and passive elements) the average number of internal nodes in each “sub-network” will be very low.

In that case the performance of any **coarse-grain** based implementation is expected to be quite bad due to the overhead associated with dealing with lists of groups of only a few nodes. In that situation, almost all nodes will be **border** and probably 99 % will be either **border** or **nearborder**. A lot of communication between "sub-networks" is then expected, which will undoubtfully degrade the performance attainable.

This problem can be seen from a different angle noting that in this case, our implementation would become very much fine-grained and we would still be using several levels of abstraction that essentially contain the same information.

On the other hand if groups are too big, that may pose some problems, because "sub-networks" will either have to wait for others, therefore leaving processors idle, or a lot of memory will be used in the checkpoints lists (if a chaotic-like control algorithm is used) and a lot of effort may be spent when rollback is needed. Clearly an optimal value will exist that minimizes the efficiency of the simulation control algorithm. This optimal value is definitely dependent on the circuit under simulation and its choice is certainly non-trivial.

3 Future Work

The partitioning strategy described, will be used in a multiprocessor implementation of **CINNAMON**. At first, we will use the "group" partitioning scheme solely. This will provide us with some insight to the optimal number and size of the sub-networks. Based on that information conclusions will be drawn and the groups recollection mechanism will be used to conclude upon the feasibility of a coarse-grain implementation.

In this implementation, the sub-networks resulting from partitioning will be kept in a linked list and simulation will proceed by having each free processor to grab a group, simulate it while a set of conditions are met, and drop it when it decides so. The decision as to when to stop simulating a group and drop it will depend on the algorithm adopted for global simulation control.

Although it is not at present possible to predict the impact of the partitioning results in the performance of the implementation, some important issues, like the size of the groups, will be studied. Also the mechanism for recollection of groups will be tested to determine whether a more elaborate algorithm is needed. The results obtained from the development of this static data-partitioning, parallel version of **CINNAMON**, will undoubtfully provide some insigth to the major problems relating to parallel circuit simulatiom.

References

[1] - Luís Vidigal et al.,

"CINNAMON: Coupled INtegration and Nodal Analysis of MOs Networks",

Proc. 23^{rd} ACM/IEEE Design Automation Conference, 1986

[2] - Luís Miguel Silveira,

"Multiprocessor Implementation of an event-driven Circuit Simulator: Evaluation of Design Options",

Esprit 1058 Report, July 1987

[3] - Luís Vidigal, Horácio Neto,

"CINNAMON: New Results and Improvements",

European Conference on Circuit Theory and Design, Paris, France, September, 1987

[4] - Randal E. Bryant,

"An Algorithm for MOS Logic Simulation",

LAMBDA, 4^{th} Quarter, 1980