RECONFIGURABLE TECHNOLOGY: AN INNOVATIVE SOLUTION FOR PARALLEL DISCRETE EVENT SIMULATION SUPPORT

C. Beaumont, P. Boronat[•*], J. Champeau, J.-M. Filloque and B. Pottier [†]

LIBr, Université de Bretagne Occidentale
BP 809, 29285 Brest, France
e-mail: <name>@univ-brest.fr

[•] Universidad Jaume I,
Apart. Correos 224, 12080 Castellón de la Plana, Spain
e-mail: boronat@vents.uji.es

ABSTRACT

Accelerating discrete event simulation can be achieved by using parallel architectures. The use of dedicated hardware is a possible alternative in some special domains like logic simulation. However, few studies have focused on general cases.

This paper presents an innovative solution using a recent hardware technology called FPGA (*Field Programmable Gate Array*), that enables dynamic synthesis of application specific hardware. Each node of an MIMD parallel machine is tightly coupled to an FPGA ring. This ring allows us to synthesize application specific *global operators* and *control or communication* circuits and complements the possibilities of the original machine on a wide application spectrum. We present the first results obtained in the simulation field with an eight node prototype.

1 INTRODUCTION

Parallel execution of discrete event simulation is a good way to reduce the processing time. Beside software studies which address general purpose parallel machine (with shared or distributed memory), few ones have focused on hardware support for PDES. Historically, logic simulation is the first domain where significant work has been undertaken. This work has been stopped for many reasons like: i) the development cost of the machine, ii) the specific usage and iii) the fast evolution of the technology of both simulated circuits and general purpose computers. More recently, research has been led to obtain solutions for general purpose simulators. Fujimoto et al. have developed the *Rollback Chip* to support memory management in optimistic simulation[3]. Reynolds et al. propose the addition of a specific active network, called PRN, to a standard MIMD machine. This network is able to compute and disseminate global information required by both optimistic and conservative PDES[7]. Their solution is attractive for coarse grain simulators. The binary-tree topology of the PRN gives good performances when the number of nodes becomes large, although the cost and complexity of interconnections between network nodes will be great. These solutions lead to efficient systems usable for a limited subset of problems.

The present paper describes a new solution using a promising hardware technology called FPGA[8]. The use of FPGA as a computing resource allows us to study the relationship between algorithms and hardware for several classes of application. The next section establishes some computing requirements proper to different classes of PDES, whereas section 3 shortly describes the hardware support and its associated software tools. Sections 4 and 5 show machine configurations for fine grain time driven simulation and synchronous coarse grain event driven simulation. We give results of experimentation and conclude with an outline of our work in progress.

2 PDES REQUIREMENTS

The domain of parallel discrete event simulation can be divided into four subdomains. FPGA ring may support critical parts of their computation.

Fine-grain time-driven PDES Let the simulated real system be decomposable into a set of small identical processes with only neighborhood interactions. Then, the model can be a state-matrix and a transition function which enables evaluation of all matrix points at the step $p+1$ from the state of their neighborhood at the step p. This model is called the *cellular automata model*[9] and can be viewed as a fine-grain time-driven simulation. Processes are synthesized in the FPGAs as *global operators* and the state-matrix is maintained in the MIMD-machine memory. Then, the machine works in SPMD mode. Section 4 gives an example of such an implementation.

Coarse-grain time-driven PDES Coarse grain time-driven simulations are necessary when processes are complex and have many interactions, i.e., there are many events in the system at each step. Processes are executed by the processors and require a synchronisation barrier at each step.

Synchronous event-driven PDES Synchronous event driven simulations require the evaluation of the minimum of all waiting events timestamps. Section 5 gives an example of an implementation.

Asynchronous event-driven PDES Asynchronous event driven simulations use either conservative or optimistic synchronization protocols. Anyway, global evaluations must be done[7], and the main one is currently computation of the minimum over distributed virtual clocks. Conservative protocols require exact evaluation of this minimum whereas optimistic ones need only approximations of the lower bound.

[*] The work of P. Boronat was done in collaboration with Telecom Bretagne has been founded by Bancaixa (grant A - 38 - IN) – Spain

[†] ArMen project is supported by French PRC-ANM and C^3 (Ministry of research and CNRS), région Bretagne, Brest municipality and ANVAR.

This minimum can be evaluated continuously by a global operator reaching local values or by a windowing mechanism as stated in an earlier paper[5]. Moreover, we have shown that it is possible to create a circuit able to evaluate minima of values belonging only to the subset of processors from which a node can receive messages. Hardware resources for such an implementation are important and not currently available on ArMen.

3 A NEW ARCHITECTURAL SUPPORT

We propose to use a general-purpose machine called ArMen. This machine is based on a modular distributed memory architecture where processors are tightly connected to an FPGA ring (see figure 1). This section gives details of the implementation.

3.1 MACHINE PRESENTATION

FPGA is a promising technology that follows the progress of VLSI integration. Conceptually, the circuits are the superposition of two planes. The upper one is a writable configuration memory, while the lower one is an active ASIC. This last plane has *Configurable Logic Blocks* (CLB), *Input/Output Blocks* (IOB) and finally a set of interconnection resources such as busses, crossbars, or programmable points. A couple of registers allows implementation of sequential logic.

Each ArMen node has a processor connected to a bank of memory and a large Xilinx FPGA. Current hardware uses Transputers and 3090/3195 FPGAs on VME-like boards. The processor loads new configuration data into its FPGA in less than 100ms. The topology of an FPGA is a CLB array with four 32-bit ports. Its north port is connected to the local address/data multiplexed system bus. East and west ports are connected to neighbors in a ring topology.

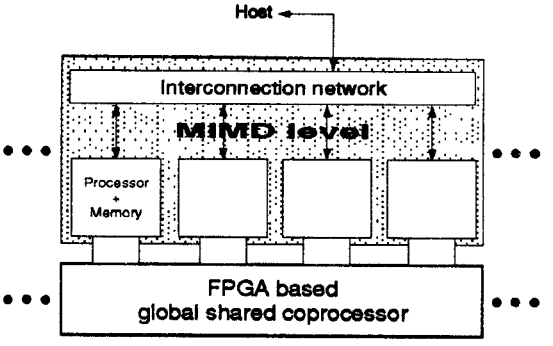

Figure 1: Principle scheme of the ArMen machine

The ArMen machine is a set of nodes interfaced to a workstation hosting a Transputer board. The machine executes the TROLLIUS operating system[2] to ease the application developments. TROLLIUS provides concurrent input/output, routing facilities and supports to load and observe processes on the nodes.

The ArMen FPGAs are local coprocessors for each node, or parts of a *global shared coprocessor* for the processor array.

3.2 SHARED COPROCESSORS

The logic layer (ring of interconnected FPGAs, also called *configurable logic layer* or CLL) of the parallel machine provides support for a large diversity of global coprocessors. They combine operative and control functions for specific tasks.

Global controllers An implicit model for global computations is based on pipelines with one stage within each node. In the pipeline mode, data is encapsulated into tokens to ensure the synchronism of their collection in the node. An automaton in the node 0 coordinates the pipelines[5].

The node processor interacts with its pipeline stage by reading or writing FPGA-based registers. Generally, local programs simply drop significant information into FPGA-based double buffered channels. Global predicates like termination detection or synchronization conditions can be computed with these global controllers. In the current implementation, tokens circulate faster than 100ns per node with inter-FPGAs asynchronous handshaking and a 20 MHz local clock for FPGAs.

Global operative units A global operative unit uses the set of local memories as a large contiguous data store, and combines the actions of the processor array with FPGA synchronous processing and micro-grain communications.

A global operation occurs in one machine cycle. During this cycle, the processor array pushes memory data to a *global operative unit*. A move from the FPGAs array to the local memories usually follows the first transfer. A linear architecture model is suitable for this kind of computation[1].

A simple global operative unit has two parts: a shift register holding a linear union of neighborhoods, and an array of small processors receiving data from the shift register. An inter-FPGA handshake protocol ensures that data lines are processed synchronously. A special node, we call the margin node, solves data dependencies for the external stripes.

Current performances of the ArMen architecture depend strongly on the processor capabilities. With Transputers, a global cycle is executed with a 1.2 μs delay, despite the operative unit complexity. This delay includes fetching a word of data from an array, writing to the FPGA and reading back the result to be pushed inside the memory.

3.3 DEVELOPMENT TOOLS

Shared coprocessors are designed using specific high level tools. They use a behavioral input description. More irregular designs, such as local accelerators, remain in the field of CAD approaches. *Cellular automata* are currently produced from a high level language called CCEL[1]. *Control circuits* are currently compiled from UCA, a language based on the UNITY formalism[4].

4 APPLICATION TO FINE GRAIN TIME DRIVEN PDES

Cellular automata are discrete dynamic models, typically fine grain parallel models. Their simulations are based on interaction between a large number of cells defined by a small set of states. The evolution is time driven: at each time step each cell computes its new state depending on the neighbors' states. This model has a large scope of simulation applications. A well known example is the game of life, an abstraction of a density regulation for a population. Cellular automata may also specify the evolution of physical systems like fluid or gas dynamics [9]. Toffoli presents cellular automata as a real alternative to the resolution of differential equations for physics modeling. In this way, Margolus proposed a specification of a gas diffusion in [9].

4.1 THE HPP-GAS MODEL

The cellular automata space domain contains gas particles. The cells' behavior is described by way of particle motions and collisions. In contrast to the classical cellular automata

neighborhood, formed by a central cell and the directly connected ones, the Margolus' neighborhood is a cell block without a specific central cell. The blocks partition the data space without overlap between the different blocks. The particles cross over to a different block because, during a time step, the block evaluation is made on two different reference grids, as shown in figure 2.

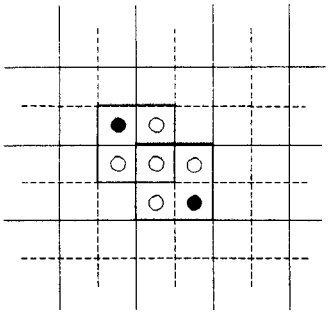

Figure 2: The two overlapping grids

In this case, the transition rule describes the evolution of a block. Figure 3 gives the motion without collisions of the particles in the model.

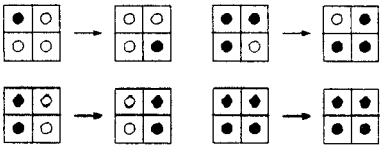

Figure 3: Particle motion

This description must be expanded with the rotated configurations of one, two and three particles. Figure 4.a shows the collision rule for two particles. They travel in opposite directions on a diagonal, starting from the other diagonal. For the other configurations, the particles act in the same way. All these rules define the HPP-gas model. They are extended by rules for particle collisions with a wall (see figure 4.b).

Figure 4: Particle collisions

4.2 IMPLEMENTATION

The implementation of this model needs four bits per cell: one for the particle, two for the reference grid specification and one to indicate the presence of a wall. The model is described and compiled with the CCEL tools after a transformation towards the classical expression of a cellular automaton. This program provides a specific FPGA configuration where a shift register contains the cell states, and an array of combinatorial functions computes the transition. For this application, the global operative unit is implemented with a 16-bit word per node, pushed in the FPGA by the processor. For one node, 4 cells are updated during a processor read/write cycle. This limitation is due to the current capacity of the FPGA.

With a data space of 350×370 cells, the time for one iteration is 64 ms using 8 nodes.

5 APPLICATION TO SYNCHRONOUS PDES

As stated earlier, many classes of global algorithms can be synthesized in the FPGAs. In this section, we present an implementation of two global operators.

The protocol implemented in order to distribute the global scheduler of sequential simulation among the processor network consists in allowing the parallel execution of all events with the same date in virtual time. We call it *phase protocol* because ready processors have to wait until all event computations for the current date (i.e., phase) are completed. Then, they can proceed to the next phase. Which will be the next date to be simulated in virtual time is determined by a global computation. This value, called *Global Virtual Time* (GVT), is the *global* minimum of all *Local Virtual Times* (LVT) (i.e., all event timestamps on each node). The main loop of this protocol is given in algorithm 1.

Algorithm 1 Main loop of our *phase*-protocol

```
1  Synchronous_DES :
2    GVT←GVT_Computation(tWakeUp_min_i);
     ▷ Global minimum computation and broadcast
3    while ( ¬ End_Simulation)
4      if ( GVT = tWakeUp_min_i ) then
5        ≪ Model evaluation;
6        Sending of generated messages;
7        Waiting for acknowledgments ≫;
8      Global_Synchronization();
       ▷ ... in order to be sure that every execution is over
       in the current time step
9      ≪ tWakeUp_min_i evaluation ≫;
       ▷ Local minimum search
10     GVT←GVT_Computation(tWakeUp_min_i);
       ▷ New global minimum computation and broadcast
```

This protocol respects the causality constraint since all processors are always executing events with the same timestamp.

In such a distributed synchronous simulation kernel, the two main global control operations are: the synchronization barrier that every processor must reach before the GVT can progress, and the calculation of the global minimum of all the LVTs in order to determine the next date to be simulated. These two global operations have been implemented in the CLL of the ArMen machine.

5.1 HARDWARE IMPLEMENTATION

The two synthesized global operators exploit the possibilities of the ArMen machine described in section 3.

In the case of the synchronization barrier, every Transputer sends a flag, set to TRUE, to its associated FPGA, indicating its wish to synchronize. Observing results from the pipelined AND-function along the FPGA ring, the automaton on node 0 will send a RESTART flag, as soon as every processor has reached the barrier. That is when it receives TRUE from the pipeline.

Considering the computation of the global minimum, we choose to implement it using a *"digit-serial"* method (see figure 5). In this case, all processors of the MIMD network have to contribute to the calculation at the same time under the control of the simulation kernel. The possibility of ArMen to switch from MIMD to SPMD mode is efficient for this kind of computation. The function implemented in an FPGA computes the minimum among three values: the local one, and the ones from the two adjacent nodes. For each node, the Transputer writes its local minimum in its local FPGA (which is seen as a part of memory addressable space),

beginning with the high order digit, and reads back the results.

After n levels of vertical pipeline, the minimum over $2n+1$ values is computed (if we have N nodes ($N > 2n+1$), we compute again the minimum on the local results until we have processed at least $N/2$ levels). It is noticeable that during the time the global minimum is computed in the FPGA ring, it is also broadcasted.

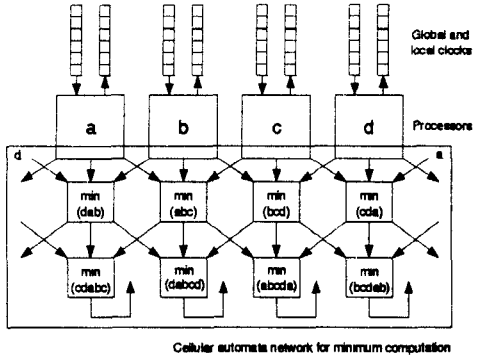

Figure 5: Digit-serial implementation of the global minimum computation and broadcast

5.2 PERFORMANCES

The performances obtained for these two global operations are summarized in table 1. One point worth being mentioned is that the number of iterated read/write operations can be reduced. Indeed, the number of accesses to the FPGA is fully dependent on the size of the minimum, on the number of nodes, and on the width of the digits. The use of larger digits (resp. 2, 4, 8 bits) reduces the number of accesses and thus the execution time of the global computation (with a ratio of 2, 4, 8, resp.). Pipelining the function also leads to such enhancements.

Number of nodes	2	4	8
Synchronization - pure soft.	4392	8481	14577
- with CLL	4	4	4
Global minimum - pure soft.	4416	10088	18160
- with CLL	110	220	440

Table 1: Execution time for global operations on the ArMen architecture (times are given in μs, global minimum is computed in "bit-serial" way on 32-bit values)

Compared to a pure software solution using the TROLLIUS operating system[2], speed-ups obtained are in order of: 1,000 to 3,500 for the synchronization barrier and 40 for a global minimum computation on four 32-bit integer values with 1-bit digits and one level of pipeline in the FPGA.

The synchronization results reflect the delay needed by the Transputer interrupt-signals. The signal transfer through an FPGA lasts indeed about 30 ns, which allows us to expect a 4 μs delay with up to 120 processors. If we consider 8-bit digits and 2 levels of pipeline, the speed-up for the GVT computation achieves 600.

6 CONCLUSION AND FURTHER WORK

The use of FPGAs as computing resource in the MIMD model opens new research areas. Lots of these are still to be investigated and we have built a parallel machine to study the relationship between algorithms and hardware for several classes of applications.

The main benefits of using reconfigurable technology in the PDES domain are possibilities for:

1. synthesizing operators for model evaluation suited to the problem or the data size.
2. synthesizing specific coprocessors able to evaluate global or partial reduction over distributed values.
3. building a *simulation machine* efficient for larger application space than previous ones, and combining the speed of hardware solutions with the flexibility of programmable ones.

We have presented two machine-configurations with "simulation-coprocessors". A fine-grain time-driven simulator with 8 nodes performs one step in 64 ms for a 350×370 cell model. A coarse-grain event-driven simulator computes each next-event timestamp in 0.4 ms. Currently, a parallel simulator is being brought to the machine and will allow performance studies on hardware accelerators for the different asynchronous protocols. In particular, we study the impact of global controller on adaptive protocols.

Other work has begun on the *bounded lag algorithm* described by Lubachevsky in [6]. A first analysis shows an important use of global operations. With the previously presented global accelerators and considering their performance, we expect that FPGAs will help to improve the simulation by an interesting ratio.

The reconfigurability enables application specific hardware optimization, and the availability of larger FPGAs in the near future allows us to consider the development of simulation coprocessors supporting both model evaluation functions and synchronization functions.

REFERENCES

[1] BOUAZZA, K., CHAMPEAU, J., NG, P., POTTIER, B., AND RUBINI, S. Implementing Cellular Automata on the ArMen Machine. In *Proceedings of the Workshop on Algorithms and Parallel VLSI Achitectures II* (Bonas, France, June 1991), P. Quinton and Y. Robert, Ed., Elsevier Science Publishers B.V., pp. 317–322.

[2] BURNS, G., RADIYA, V., DAOUD, R., AND MACHIRAJU, R. All About TROLLIUS. *Occam User Group Newsletter* (July 1990), 55–70.

[3] BUZZEL, C., ROBB, M., AND FUJIMOTO, R. Modular VME RollBack Hardware for Time Warp. In *Proceedings of the SCS multi-conference on Distributed Simulation* (San Diégo, USA, Jan. 1990), SCS, pp. 153–156.

[4] DHAUSSY, P., FILLOQUE, J.-M., POTTIER, B., AND RUBINI, S. Global Control Synthesis for an MIMD/FPGA Machine. In *IEEE Workshop on FPGAs for custom computing machines* (Napa, California, Apr. 1994), I. C. S: Press, Ed.

[5] FILLOQUE, J.-M., GAUTRIN, E., AND POTTIER, B. Efficient Global Computation on a Processor Network with Programmable Logic. In *Proceedings of PARLE'91* (Eindhoven, NL, June 1991), no. 505 in LNCS, Springer-Verlag, pp. 55–63.

[6] LUBACHEVSKY, B. Synchronization Barrier and Related Tools for Shared Memory Parallel Programming,. In *Proceedings of Intl. Conference on Parallel Processing,* (Pen State, USA, Aug. 1989), pp. "II–175–179".

[7] REYNOLDS, P. Efficient Framework for Parallel Simulations. In *Proceedings of the SCS multi-conference on Advances in Parallel and Distributed Simulation* (Anaheim, USA, Jan. 1991), SCS, pp. 167–174.

[8] ROSE, J., GAMAL, A. E., AND SANGIOVANNI-VINCENTELLI, A. Architecture of field-programmable gate arrays. *Proceedings of the IEEE 81*, 7 (July 1993).

[9] TOFFOLI, T., AND MARGOLUS, N. *Cellular automata machines.* MIT Press, 1987.