Acceleration of Multi-agent Simulation on FPGAs

Lintao Cui, Jing Chen, Yu Hu
Department of Electrical
and Computer Engineering
University of Alberta
Alberta, Canada
Email: {lintao, jc14, bryanhu}@ece.ualberta.ca

Jinjun Xiong
IBM T.J. Watson Research Center
New York, USA
Email: jinjun.ucla@gmail.com

Zhe Feng, Lei He
Electrical Engineering Department
University of California, Los Angeles
California, USA
Email: feng07@ucla.edu
lhe@ee.ucla.edu

***Abstract*—Multi-agent simulation (MAS) is a widely used paradigm for modeling and simulating real world complex system, ranging from ant colony foraging to online trading. The performance of existing MAS software, however, suffers when simulating massive-scale multi-agent systems on traditional serial processing processors. In this paper, we propose an FPGA-based framework for massive-scale grid-based MAS. Memory interleaving, parallel tasks partition, and computing pipeline are adopted to improve system throughput. A classical MAS benchmark, Conway's Game of Life, is used as a case study to illustrate how to map grid-based models to our MAS framework. We implemented it on a Xilinx Virtex-5 FPGA board and achieved a speedup of 290x with two million agents, compared to the C implementation.**

I. Introduction

A multi-agent system is comprised of multiple interactive and intelligent agents, and each agent makes its own decision based on the current situation and a set of rules. Like ant colony and bird flock in real world, complex or unanticipated group behavior patterns could be generated even by simple rules. The famous ant colony optimization was inspired by inspecting ant colony foraging and is effective in discrete optimization related to swarm, which demonstrates the significance of exploring multi-agent systems.

The repetitive and complicated interaction among agents is out of the range of traditional equation-based modeling, which lacks the insight of micro-level interaction among agents. Multi-agent simulation is a computational model to simulate such systems. It utilizes a bottom-up approach to simulate the interaction among multiple agents at the micro-level, to predict the potential appearance of complex group behavior at the macro-level. It has been adopted in a wide range of areas, including animal group behavior analysis [1], social network analysis [2], influenza prevention [3], etc.

A number of MAS software have been developed and are available in public domain, which provide a programmable platform for simulating multi-agent systems, such as StarLogo [4] and SWARM [5], all running on a general-purpose CPU. Those tools are highly serial and lack of parallel computing. This is particularly troublesome in cases like emerging infectious disease prevention which usually involves a large number of agents and need almost an instant result. For example, New York City has a population of nearly twenty million, but the capacity of current desktop MAS can handle at most several thousands [6]. Therefore, a high performance acceleration platform based on parallel execution is desired to fully exploit MAS capabilities. FPGA (field programmable gate array)-based approach has shown great potential in terms of energy efficiency, and highly parallel architecture [7].

In this paper, we propose a novel FPGA-based acceleration architecture to speedup grid-based MAS. Memory interleaving is utilized to form a flexible memory architecture and accelerates single agent's communication significantly. Parallel tasks partition and pipeline are adopted for parallel multiple agents computing to improve the throughput. We analyze the classical MAS benchmark, Conway's Game of Life (CGL), to demonstrate how to map the grid-based models to our FPGA-based architecture. CGL is implemented on a Xilinx Virtex-5 board running at 100MHz and we achieve 290x speedups compared to the C implementation, running on an AMD Athlon 2.9 GHz Quad-core CPU with 6 GB RAM.

The remainder of this paper is organized as follows. Section II presents the basic concepts of MAS and a brief survey of related work on acceleration of MAS. In section III, we narrow down our research scope to grid-based MAS and introduce our FPGA-based approach for accelerating grid-based models. Section IV shows how to accelerate CGL using our approach and discusses the experimental result. In Section V, we conclude the paper with future research directions. To the best of our knowledge, this work is the first in-depth study of accelerating MAS on FPGA.

II. Preliminaries

A. Concept

Typically in MAS, there are a number of interactive and intelligent agents, which move around at an environment, and a set of rules governing agents' behaviors. Based on the properties of the environment, MAS is classified as two types.

1) *Grid-based MAS:* The environment is modeled by a multi-dimensional grid consisting of lots of cells, where mobile agents move around. Most of MAS fall in this area, like cellular automata, and molecular dynamics.

2) *Graph-based MAS:* The environment is represented by a graph of relationship network with nodes, connected to other nodes through relational edges. Social network is graph-based MAS.

978-0-7695-4529-5/11 $26.00 © 2011 IEEE
DOI 10.1109/FPL.2011.92

Due to the fact that grid-based MAS covers the majority of MAS applications, our work currently focuses on how to build a framework for grid-based models on FPGA.

A typical MAS consists of the following three core components.

1) *Environment:* Environment forms the background for MAS. Typically, there are a number of property values stored, representing certain attributes, such as temperature. The movement of agents within a grid can be modeled by copying the state attributes of an agent from the source cell into the destination cell.
2) *Intelligent Agents:* Agent is the basic unit in MAS and performs actions based on a set of rules and current environment. Typically, there is no controlling agent within a MAS, and each agent carries out the following tasks in each step:
   - Fetch the agent state
   - Communicate with local environment and neighbors
   - Rule compute and make decision
   - Update the agent state
3) *Communication:* MAS typically exhibits intensive communication and interaction between agents and environment or agents and their neighbors.

B. High Performance Computing for MAS

Recent researches have attempted to accelerate MAS on parallel and distributed platform, such as GPU (graphic processing unit), and computing cluster and grid.

1) MAS acceleration on GPU: GPU is very effective at manipulating computer graphics. Its highly parallel structure has attracted some efforts in high performance MAS. FLAME is a GPU-based framework, designed for parallel MAS. It has achieved a speedup up to 250 times in contrast with a single CPU implementation [8]. Though the speedup is significant, the disadvantage of GPU-based architecture is its large power consumption, which significantly increases the operation cost.

2) MAS acceleration on Cluster: Some other MAS researches favor the computing cluster. IBM Tokyo Research Lab has developed a large-scale MAS framework on BlueGene, a multi-node supercomputer [9]. However, the power of cluster-based computing resides in high speed computing with non-interactive workloads. Therefore, the bottleneck of communication bandwidth between computers is encountered when simulating communication-intensive large-scale applications, which is common in MAS.

3) MAS acceleration on FPGA: Few work has been done to accelerate MAS on FPGA. The only one known to us is the parallel implementation of SOARS in which FPGA is used as an alternative for multi-processor architecture without significant results published yet [10]. Actually, FPGA is a very promising platform for high performance computing with highly parallel and flexible architecture [7].

In this paper, we propose an FPGA-based framework to accelerate grid-based MAS. We implemented CGL as a demo and achieved a significant 290x performance improvement.

III. Acceleration for Grid-based MAS

Here we focus on grid-based models, which cover the majority of MAS applications. More specifically, we study the models featuring the following characteristics:

- Environment is grid-based, consisting of lots of cells
- Rule computation demands a cluster of local agents
- Communication happens within a neighborhood.

In this section, we first present how to accelerate a single agent using memory interleaving and then the multiple agents acceleration through parallel task partition and pipeline.

A. Single Agent Acceleration

Based on the second premise, updating an agent's state, which is stored in a cell, depends on a cluster of neighboring cells. Traditionally we need to access those cells one by one. The system performance will benefit a lot if we can access those neighboring cells simultaneously. Here, we use memory interleaving [11] to achieve concurrent access to all of the cells in one memory accessing cycle.

Today's high-volume FPGA chips usually hold hundreds of configurable Block RAMs (BRAM). The configurable features, like accessing ports, data width and memory address, enable the possibility of designing a flexible high performance memory architecture at no cost.

Memory interleaving takes the advantage of those BRAMs. The basic idea is to scatter a cluster of multiple data into different BRAMs based on the LSBs (least significant bits) of their coordinates so that we can access all of them at one time.

For example, in a two-dimensional grid, to update the cell A, a total number of $N \times M$ cells are required, defined as a cluster as shown in Fig. 1. In this case, the n-LSBs in coordinate X and m-LSBs in coordinate y are used, leading to the value $N = 2^n$ and $M = 2^m$. Thus, we can map those cells into different BRAMs, numbered from one to $N \times M$

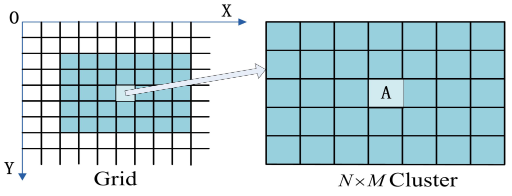

Fig. 1. A $N \times M$ Cluster. N is the number of cells along X-axis, M is the number of cells along Y-axis.

TABLE I
Mapping data to BRAMs

X n-LSBs	***Y m-LSBs***	***BRAM NO.***
0	0	1
0	1	2
⋮	⋮	⋮
0	M-1	M
1	0	M+1
⋮	⋮	⋮
N-1	M-1	$N \times M$

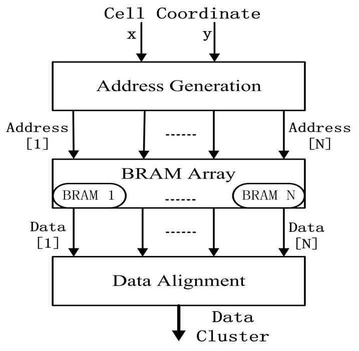

Fig. 2. Memory accessing with N BRAMs

and forming a BRAM array, as listed in Table I. Therefore, those cells can be concurrently accessed. For a cluster with N cells, we can improve the memory reading bandwidth by N-1. The improvement is bigger with larger cluster size.

The general memory accessing structure using memory interleaving is shown in Fig. 2. The BRAMs store the data based on the mapping rule detailed above. When accessing the memory to update a cell (x,y), the address generation module will generate N addresses required for the corresponding BRAMs based on the input coordinates. The output data will be aligned by the data alignment module to form the cluster pattern as defined and prepared for further processing.

B. Multiple Agents Acceleration

In this subsection, we present the acceleration of multiple agents. First, given a grid representing agent states, how to partition it into parallel sub-grids. Second, for a sub-grid, how to improve the computing throughput by pipelining its processing flow.

1) Parallel Tasks Partition: As shown in Fig. 3, given a grid, we can evenly partition it into non-overlapping sub-grids along the Y-axis and each of them will be stored in the corresponding BRAM array using memory interleaving. BRAM arrays are allocated to the corresponding processing elements (PE) for parallel processing. PEs are independent computing modules responsible for the rules computation.

Clearly, communication across BRAM arrays is necessary, because the updating of a bordering cell depends on its neighboring cells stored at other BRAM arrays. Since we partition the whole grid along its Y-axis, communication across borders could be achieved by checking the MSBs (most significant bits) of its Y coordinates. For example, if we evenly partition the grid into four sub-grids along Y-axis, a cell can be accessed by first checking the two MSBs of its Y coordinates to determine which BRAM array this cell is stored in and further to decide at which BRAM it is located based on the LSBs of its coordinate. The BRAMs can be configured as dual-port memories, which thus support the concurrent memory accessing from two PEs. Therefore, we can complete the bordering communication without redundant data storage.

By dividing a grid into N sub-grids, we can improve the

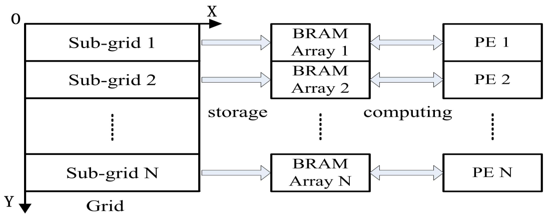

Fig. 3. Grid partition with N sub-grids

system throughput by N-1. The number of partition is only limited to the FPGA resources.

2) Computing Pipeline: Pipeline can improve the system computation throughput when processing a stream of data. For a sub-grid and its corresponding BRAM array and PE, the processing flow roughly takes several different stages: fetch data, rule computing, and update grid. Therefore, we can implement a three-stage pipeline to accelerate the performance. Given a specific application, fetch data and rule computing stages can be divided into multiple stages based on the maximal combinational delay to increase the system operation frequency.

IV. Case Study on Grid-based Models

In this section, we illustrate how to map CGL, which meets the three premises, to our grid-based MAS framework.

A. Conway's Game of Life

CGL is a cellular automation devised by John. H. Conway. It is a zero-player game, requiring no further input, whose evolution is determined by its initial state. Its environment is a toroidal two-dimensional gird of cells, each of which is in one of two possible states, live or dead. Every cell interacts with its eight neighbors as shown in Fig. 4. In each step , the following rules apply.

- Live cell with fewer than two live neighbors dies.
- Live cell with two or three live neighbors lives.
- Live cell with more than three live neighbors dies.
- Dead cell with exactly three live neighbors becomes alive.

B. Mapping Game of Life for Acceleration

In CGL, the environment and cell agents are the grids themselves. As shown in Fig. 4, to update the states of cell E, a cluster of nine cell states is required. Thus, each BRAM array should be consisted of nine BRAMs, numbered from 1 to 9. Cells are mapped into BRAMs as shown in Table II.

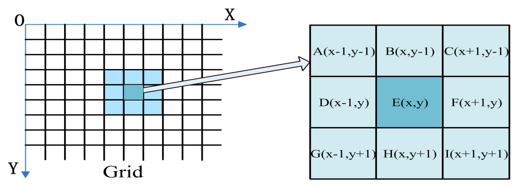

Fig. 4. A cluster of 9 cells in CGL

TABLE II
MAPPING DATA INTO BRAMS

x LSB	*y LSB*	*BRAM NO.*
0	0	1
0	1	2
0	2	3
1	0	4
1	1	5
1	2	6
2	0	7
2	1	8
2	2	9

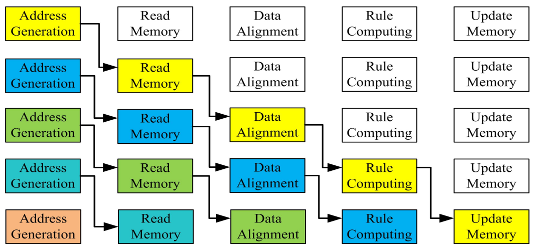

Fig. 5. A five-stage pipeline for CGL

Given a certain size of grid, we can partition it into multiple sub-grids, stored by the corresponding BRAM arrays and processed by the corresponding PEs. On a Xilinx Virtex-5 FPGA, which contains 148 BRAMs, we can implement eight PEs and eight BRAM arrays, consuming 128 BRAMs. And thus the grid is divided into eight sub-grids and eight PEs are implemented.

The fetch data stage can be further divided into three stages: address generation, memory reading, and data alignment. The rule computation stage in CGL is simple and takes only one stage. Therefore, we can implement a five-stage pipeline to manipulate a sub-grid: address generation, read memory, data alignment, rule computing, and update memory as shown in Fig. 5.

We implemented CGL on a Xilinx Virtex-5 board. The synthesis result by ISE shows the LUT (look up table) utilization is only 12% with maximal frequency at 136 MHz while the BRAM utilization is up to 86%. Therefore, it is possible to further improve system performance by constructing more BRAM arrays through distributed BRAMs which are generated by LUTs.

We set the system frequency at 100 MHz, and compared the experimental result with a C implementation running on an AMD Athlon 2.9 GHz Quad-core CPU with 6GB RAM. We ran the experiments with four different grid sizes: 10000, 100000, 1 million and two million. The speedup result is shown in Fig. 6.

The result shows that we are able to achieve a maximal speedup of 290 when the grid size is two million. What's more, the speedup does not decrease when the size of grid increases, in fact it is directly related to the number of available BRAM arrays and PEs. We can get a better result by combining embedded BRAMs and distributed BRAMs to get a maximal

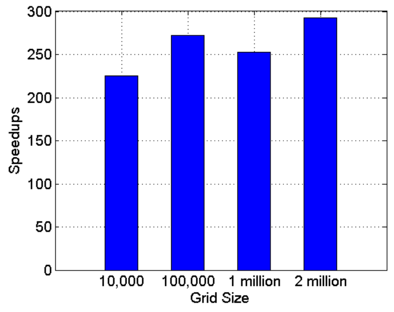

Fig. 6. Experimental result with four different sizes of grid

parallelization, which is actually resources tradeoff.

V. Conclusions and Future Work

In this paper, we have introduced a general FPGA-based framework for grid-based MAS, which could be generalized to MAS applications meeting our premises. The experimental result shows that our architecture can accelerate the classical Game of Life by 290x when using a Xilinx Virtex-5 board running at 100 MHz, compared to C implementation.

In the near future, we will extend our FPGA-based framework to a more general grid-based MAS by studying more applications and models, and develop automatic mapping and code-generation tools to produce high-quality parallel code and appropriate architecture across vastly different grid-based models. We also plan to develop a general FPGA-based framework for graph-based MAS, and thus we can test if FPGA could accelerate other kinds of MAS at a high performance.

References

[1] E. Merelli *et al.*, "Agents in bioinformatics, computational and systems biology," *Briefings in Bioinformatics*, 2007.
[2] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. Suppl 3, p. 7280, 2002.
[3] H. Nathaniel *et al.*, "The virtue of virtuality: the promise of agent-based epidemic modeling," *Laboratory and clinical medicine*, 2008.
[4] StarLogo. http://education.mit.edu/starlogo.
[5] Swarm. http://www.swarm.org/index.php/Main_Page.
[6] M. Lysenko and R. DSouza, "A framework for megascale agent based model simulations on graphics processing units," *Journal of Artificial Societies and Social Simulation*, vol. 11, no. 4, p. 10, 2008.
[7] M. Gokhale, J. Cohen, A. Yoo, *et al.*, "Hardware technologies for high-performance data-intensive computing," *Computer*, vol. 41, no. 4, pp. 60–68, 2008.
[8] P. Richmond, S. Coakley, and D. Romano, "A high performance agent based modelling framework on graphics card hardware with CUDA," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 2009, pp. 1125–1126.
[9] T. Takahashi and H. Mizuta, "Efficient agent-based simulation framework for multi-node supercomputers," in *Proceedings of Winter Simulation Conference*, 2006, pp. 919–925.
[10] H. Tanuma, H. Deguchi, and T. Shimizu, "Hardware Implementation of Parallel SOARS using FPGA based Multiprocessor Architecture," *Agent-Based Approaches in Economic and Social Complex Systems IV*, pp. 199–206, 2007.
[11] T. VanCourt and M. Herbordt, "Application-Specific Memory Interleaving for FPGA-Based Grid Computations: A General Design Technique," in *Field Programmable Logic and Applications, International Conference on*, 2006, pp. 1–7.