Using Field-Programmable Gate Arrays for Learning Non Player Characters

Christopher Cichiwskyj
Embedded Systems
University Duisburg-Essen
47048 Duisburg
Germany
christopher.cichiwskyj@uni-due.de

Gregor Schiele
Embedded Systems
University Duisburg-Essen
47048 Duisburg
Germany
gregor.schiele@uni-due.de

ABSTRACT

In this paper we present ongoing work on how to use Field-Programmable Gate Arrays to increase the number of concurrent non player characters in large scale interactive virtual worlds. We employ reinforcement learning combined with artificial neural networks to allow the simulated characters to learn from previous engagements with players. Our simulations show achievable performance gains of several orders of magnitude compared to a CPU-based solution.

CCS Concepts

•Computing methodologies → Multi-agent reinforcement learning; •Hardware → Hardware accelerators;

Keywords

FPGA, Reinforcement Learning, Massively Multiuser Virtual Environments

1. INTRODUCTION

A field programmable gate array (FPGA) is an integrated circuit consisting of programmable logic blocks that can be interconnected dynamically by a programmer. This essentially allows the programmer to develop custom hardware that is highly optimised for a specific application. The FPGA can be reprogrammed with a new circuitry layout at any time [1]. If the application changes, its custom made hardware can be adapted accordingly. Additionally, FPGAs offer instruction level parallelism, which allows the programmer to create highly parallel computation structures on a very low hardware level. FPGAs can be used in a variety of different areas, e.g. cryptography [2] or speech recognition [3].

We are convinced that FPGAs have a huge potential for Massively Multiuser Virtual Environments (MMVE) because MMVEs contain a lot of parallel, situation-dependent tasks. So far only a very limited amount of work has been done in that regard [4][5]. In this paper we present our current work in this area. We use FPGAs to increase the number of concurrent learning non player characters (NPCs) in an MMVE. This is one example how FPGAs can help to overcome current restrictions of MMVEs, namely the necessity of spreading NPCs in the virtual world.

Our paper is structured as follows. First we discuss our idea how FPGAs can be used to dynamically adapt servers to different overload situations. Then we present how this general idea can be applied to the artificial intelligence (AI) engine. We show some preliminary performance measurements and conclude with a short outlook on next steps.

MMVE'16, May 12 2016, Klagenfurt, Austria

ISBN 978-1-4503-4358-9/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2910659.2910662

2. FPGA-BASED MMVE SERVERS

To realise huge simulated virtual environments, MMVE systems distribute the execution workload on many interconnected servers. This approach works as long as the workload is spread evenly in the virtual world. If the workload in a small area gets too high, e.g. because of a battle involving hundreds of players and NPCs, a single server is overloaded. Further distribution of the workload is usually not possible due to the necessary level of inter-server synchronisation.

FPGAs can help solving this issue. An FPGA can be programmed to execute key MMVE functions in hardware, i.e. with very high speed. Thus, a single server with an FPGA can handle much more workload. At the same time we can reprogram the FPGA at runtime. By analysing the overload situation, we can determine its causes and then deploy the most needed key function(s) on the FPGA. As an example, if a server is overloaded because it has to simulate too many NPCs, then parts of the AI engine can be moved to the FPGA (see Figure 1). If many thousands of starships are moving close-by in a space battle, parts of the physics engine may be deployed on the FPGA in a similar fashion.

3. EXAMPLE: LEARNING NPCS

In our current work we focus on the AI engine of an MMVE. To create the illusion of a living world, a typical MMVE is populated by many non player characters (NPCs), e.g. acting as citizens of a virtual city or soldiers in a virtual army. The perceived intelligence of these NPCs can influence the immersion and long term motivation of players. If NPCs learn from previous encounters with players and adapt their behaviour accordingly, then players have to come up with new strategies and thus can replay the same scenario more often. This reduces the pressure on MMVE providers to create new content and thus reduces cost. However, suitable

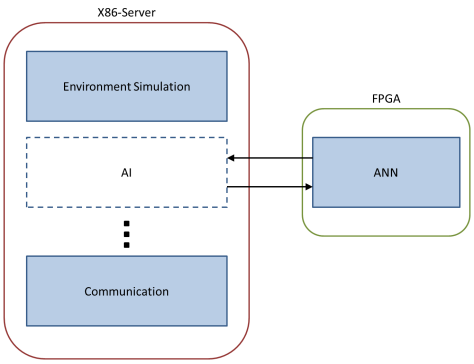

Figure 1: Possible software architecture on hardware with FPGA as accelerator containing an artificial neural network (ANN)

machine learning techniques, like reinforcement learning, are computationally complex and are thus usually infeasible to use for simulating large numbers of NPCs on typical server hardware. Reinforcement learning (RL) [6] is based on the idea that entities learn from their previous experiences. In a given situation, an entity decides to perform an action and receives a reward, based on the effect of this action in the current situation. As an example, if an NPC decides to attack a player and is killed, it receives a negative reward. It learns that attacking in this situation is a bad decision and will try to avoid this in the future. If the NPC is able to sell an item with a profit, it will receive a positive reward, indicating that this action is beneficiary.

To implement RL efficiently a feed-forward artificial neural network (ANN) [7] can be used to store past experiences of NPCs. Such an ANN consists of connected neurons that are grouped into layers. Each layer is only connected to the layer before and after itself. No loops are permitted. This type of ANN is easily parallelised, but on an x86-architecture CPU we only receive a speed up in relation to the number of processor cores. With an FPGA, the situation is entirely different. Due to its instruction level parallelism, the FPGA can calculate all neurons in one layer in parallel in a single clock cycle. The calculation of a complete ANN takes the clock speed times the number of layers. The number of neurons per layer is irrelevant for the calculation speed. This means that the size of a layer is not limiting the calculation speed and is only limited by the physical area the FPGA provided for design.

4. PRELIMINARY RESULTS

We are currently developing an implementation of such ANNs for FPGAs. Our target hardware is a heterogeneous processor server provided by the FiPS project [8]. To evaluate the possible performance gains, we conducted first simulations with the Xilinx Vivado Design Suite. In this simulation we compared an ANN implementation for Xilinx Zynq XC7Z045 FPGAs with a reference Java implementation run on an Intel Core i7-4710MQ. As shown in Table 1, the FPGA-based implementation achieves a speed up of a factor of 1000. However, in this scenario the FPGA size was sufficient to fully parallelise all neurons per layer in the ANN. Also, the results do not take into account possible performance losses due to necessary synchronisations with other MMVE server components that are executed on the x86-CPU.

	CPU time (s)	FPGA time (s)
ANN 5x5	$1.03 \cdot 10^{-4}$	$2.00 \cdot 10^{-7}$
ANN 5x20	$1.24 \cdot 10^{-4}$	$8.00 \cdot 10^{-7}$
ANN 20x50	$2.11 \cdot 10^{-4}$	$(2.00 \cdot 10^{-6})$

Table 1: Comparison of execution time of different ANNs on an Intel Core i7-4710MQ and a Xilinx Zynq XC7Z045 FPGA (simulated values) in seconds

5. CONCLUSION

Our results so far are promising. By deploying key parts of the AI engine on an FPGA, we can speed up the computation of learning NPCs by several orders of magnitude. We believe that this will make it feasible to use learning NPCs in real MMVE systems and get beyond simple rule-based approaches. Clearly, this is work in progress and much more research is needed. Our system implementation must be finished and evaluated on real FPGA hardware. We also need to analyse interdependencies with other MMVE server components and their impact on performance. In future work we plan to extend our work to use multiple interconnected FPGAs and to deploy other engine functions on them.

6. ACKNOWLEDGMENTS

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 609757.

7. REFERENCES

[1] Stephen Trimberger. A reprogrammable gate array and applications. *Proceedings of the IEEE*, 81(7):1030–1041, Jul 1993.

[2] M. Nawari, H. Ahmed, A. Hamid, and M. Elkhidir. Fpga based implementation of elliptic curve cryptography. In *Computer Networks and Information Security (WSCNIS), 2015 World Symposium on*, pages 1–8, Sept 2015.

[3] J.W. Orillo, R. Yap, and E. Sybingco. Improved noise robust automatic speech recognition system with spectral subtraction and minimum statistics algorithm implemented in fpga. In *TENCON 2012 - 2012 IEEE Region 10 Conference*, pages 1–6, Nov 2012.

[4] Jonathan Sieber. Implementing the nintendo entertainment system on a FPGA, 2013.

[5] Nintendo ninja - a hardware-based FPGA AI for super mario bros. http://www.nintendoninja.com/. Accessed: 2016-02-18.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[7] S. Agatonovic-Kustrin and R. Beresford. Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis*, 22(5):717–727, 2000.

[8] Mario Porrmann. D1.2.1 - specification of the hardware architecture. Public deliverable, The FiPS Project (FP7-ICT-2013- 10 (609757)), 2014.