

Multiple platforms: Issues of porting Agent-Based Simulation from Grids to Graphics cards

[Extended Abstract]

Mariam Kiran
School of Electrical Engineering and
Computer Science
University of Bradford
Bradford, UK
m.kiran@bradford.ac.uk

ABSTRACT

Agent-based modelling and simulation techniques are computationally demanding, allowing simulation of large complex models which can scale up and have high memory processing and storage demands. Simulating biological models formed by individual cells are particularly favoured on these kinds of modelling techniques. FLAME (Flexible Large scale Modelling Environment) framework is one example of a framework which allows modellers to build serial and parallelized versions of the models by simply adding extra flags during execution. In this paper, we discuss the issues faced when porting the same model developed for simulation from FLAME-HPC to graphics card version FLAME-GPU, highlighting the advantages and disadvantages from the modelling and simulation perspectives. The paper discusses the experience and issues faced when models had to be essentially rewritten for a GPU version, enhancing simulation time but limiting model complexity in the process.

Categories and Subject Descriptors

C.1.4 [Parallel Architectures]: Distributed computing I.6
[Simulation and Modelling] architectures and languages

General Terms

Performance, Design, Experimentation, Modelling.

Keywords

FLAME, FLAME GPU, Biological modelling

1. INTRODUCTION

Modelling behavior of complex systems is an emergent science which demonstrates the complex and social behavior of large scale communities working together in real world scenarios. Examples of ant colonies, social networks and even bird flocking is referred to as complex systems consisting of individuals interacting together to produce emergent behavior (Figure 1). Simulating complex models is a cumbersome task involving massive data processing, storage issues and pattern recognition of behaviors within the system. Depending on the complexity of the model, some simulations may take days or even weeks to finish processing. Sometimes these would abandon due to low memory or loss of processing power of the infrastructure underneath. Past modelling techniques which involve large-scale system modelling have used high performance computing grids and GPU cards to quickly process large complex equations for multi-massive variables to produce emergent solutions [3, 4, 6] to predict how systems behave. Agent-based modelling (ABM), also known as individual-based modelling, is a technique which best models complex systems by modelling the individual interacting elements rather than a whole system and serves as an alternative to conventional differential

equation which models the complete system. This approach allows a bottom-up approach of generating behaviour from the bottom up concentrating on the individual interacting units which are given clear defined rules and allowed to simulate. The produced emergent pattern of system behaviour, can then be studied to test and understand the behaviour of complex systems which is otherwise not possible from studying these systems from an outside view.

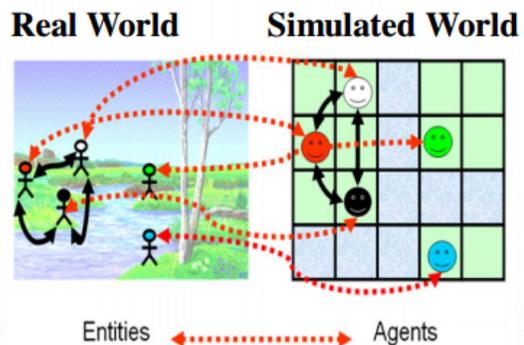


Figure 1. Mapping real world complexity to simulated environment. C.f. [1]

Due to the complexity of these models and the time it takes to simulate them, researchers often use high performance computing grids to reduce the time it takes to simulate and analyse the results. Working with economists and biologists, computer scientists developed the FLAME framework which can allow non computer scientists to easily write their models in an easy to understand language specification, which could then automatically produce parallelizable code to execute on grids. Initially developed for HPC grids, the framework was adapted for executing on Nvidia graphics cards to allow simulations to run faster, improving performance upto 80% on a single machine [8]. However this capability comes with a cost and technical limitations on the models themselves. This paper aims to discuss these experiences from a modeller's perspective, highlighting the challenges of multiple platforms being used to simulate the same complex model across two platforms, to discuss the following research questions:

RQ1: Is it feasible to have unified modelling language to define a model abstraction, to allow execution on multiple platforms?

RQ2: Is it useful to define favourable architectures for particular models in advance?

RQ3: What are the tradeoff when using multiple platforms for simulation?

RQ4: Challenges faced by modelers in writing, simulating and testing the models.

2. FLAME FRAMEWORK

The Flexible Large-scale Agent Modelling Environment (FLAME) (www.flame.ac.uk) is a framework which is specially written for simulating large populations of agents or individual software elements on parallel architecture. Till date, there are various agent-modelling frameworks such as Repast, Jade developed by IBM and Netlogo being extensively used by the social science community, some of these allow only serial executions requiring the model to be specially rewritten for parallel execution.

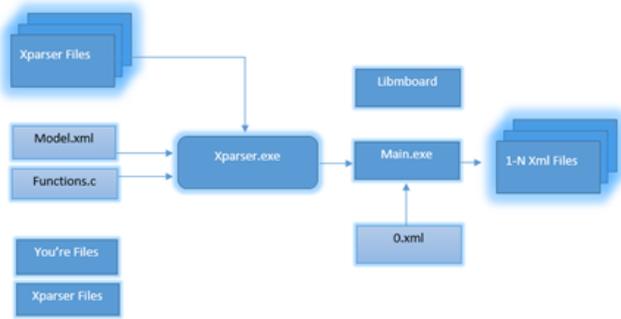


Figure 2. Block diagram of FLAME framework

FLAME acts as an enabling tool to create agent-based models. The agents are based on the extended state machine (X-machine) methodology which allows definition of state machines to be equipped with memory and communicating messages. Transitions between functions of each X-machine is determined by the memory state and messages being read by the X-machine agent (Figure 3). These agents can be defined by modellers, from any domain, as cells in a biological model or as banks and firms in an economic model. Different agents interact using messages being read/written through the transition functions and message boards.

Agent-based model engineering orients towards how the requirements and the system are represented. Use of formal specification methods and different processes to capture internal and external concepts of multi-agent systems has been highly researched [10, 11]. Various approaches have been discussed on how the behavior of the system can be represented using semantic approaches such as model checking [12].

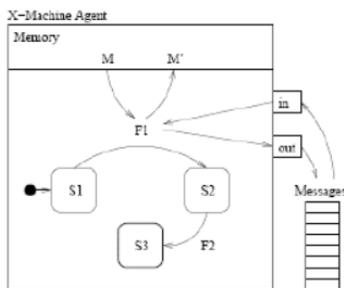


Figure 3. X-Machine agent

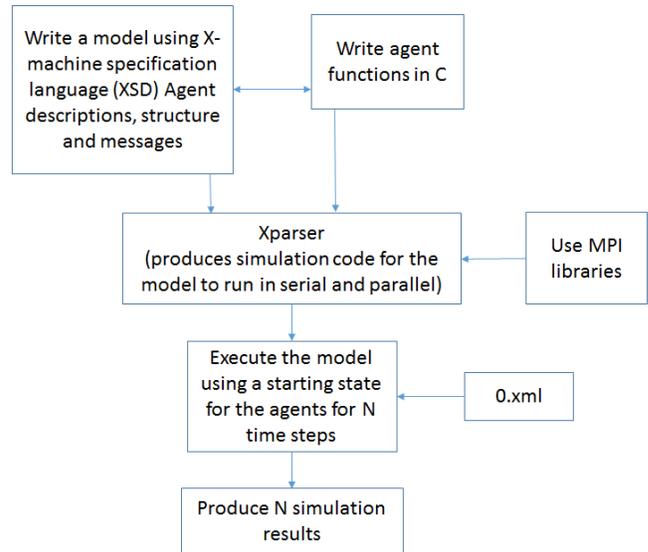


Figure 4. Block diagram of FLAME working.

Figure 4 describes how models are written in a descriptive XML notation, which is then fed into a parser. The parser produces automatic parallelizable code which is ready to be run over multiple processors of a grid. The same parser can also run the code on GPU cards. Simulation results are produced on disk to be analysed later.

ABMs are based on the principles of cellular automata updates. Popular agent-based frameworks allow synchronizations using update schedulers such as used in MASON, REPAST, and SWARM. FLAME uses initial synchronization calculations using the definition of the model from the definition language. The parallelization is then done using MPI interacting with C functions to allow agents to be distributed across multiple processors synchronization every time they write and read to message boards. FLAME prevents any asynchronous agent update as in cellular automata models.

2.1 Porting on GPU

The GPU version of FLAME shares the same principles such as using the same specification language (XML) as an initial description of the models. However the agent functions are written in C++ and interface with CUDA libraries such that they can be processed on the machine's local graphics cards. The models showed an 80% improvement in simulation time and allowed real-time and 3D rendering of results which the simulations are running [8]. Each agent would thus become an independent thread performing its functions wrapped by the GPU kernel.

3. PORTING PROBLEMS

Although both versions of FLAME use similar methodologies and description languages, a number of changes had to be introduced into the model itself, prior to the parser step, to ensure it runs on the GPU. Upon inspection, most of these changes were due to the basic capability of the HPC grid versus the GPU cards. These changes have been discussed below to highlight some issues raised by portability from a modellers point of view:

3.1.1 Writing the Agents

From the modeller's perspective, implementing the system in FLAME in general was quite straight forward provided certain rules were being follows. Due to synchronization nature of the

framework which are predefined before the simulation runs, FLAME does not allow agent behavior to contain loops which may cause it to go back to a previous states. Similarly communications are decided at specific steps before the code is parallelized to ensure synchronization points (when the message boards are locked for reading and writing) to prevent any discordances in the data of the messages read, following a step-by-step progression in the model.

3.1.1.1 Pre-allocation of Agent memory

The grid FLAME version allowed memory to be allocated dynamically as the simulation runs. This allowed complex agent memory such as using dynamic arrays or linked lists to be generated as the model runs on the grid. This however, is not the same on GPUs. The GPU needs for all memory to be allocated in advance before the simulation starts, due to the manner in which the agents are executed as threads on the cards. This means that all dynamic arrays in a model had to be changed to static arrays of specific sizes, which caused considerable changes on the model reducing some of the model complexity by limiting the memory size of the agents.

In addition to the strict memory sizes, the grid version allows memory data structures of multiple data types to be created. This capability was not exhibited by FLAME GPU, allowing only a specific type of variables of fixed lengths data types. This raises issues if there are certain data structures used in the model which act as records of multiple datasets used by the agent such as a product inventory or biological rules. The cell model [9] was changed from a data structure into a 1D array and stored in memory where agents could globally read it serially. This caused a considerable rewriting of the model itself, however it did reduce the potential processing time of the model a data was being access serially rather than as dynamic data structures in the HPC version.

3.1.1.2 Message Communication between agents

In the grid version of FLAME, signals can be passed between agents as messages with each agent creating multiple messages as needed. FLAME GPU limits this to only one message per function to be created which limited the amount of information that was possible with one function. The model has to be broken down to allow multiple function transitions within one function to allow messages to be communicated making the model quite complex or expanding the message at times to create much more information than previously designed. This constraint was introduced due to the manner in which the agent threads communicate allowing only single message to be synchronized per function.

3.1.1.3 Simpler versus Complex

Although both versions grid and GPU are suitable for simulation, the GPU version seems more suitable for simpler model executions with agents with basic memory allowing much faster simulations as compared to the grid version. This makes it suitable for game based simulations to quickly visualize the results quickly and where data analysis is not an extensive requirement. However with more complex models such as the model of the complete European economy, with 15 different agent types with increasing complexity and multiple interactions, such models could not be processed on GPU cards.

3.1.1.4 Looping through Messages

Following is an example of looping through a message list as defined in FLAME HPC, which allows an agent to read through all messages in the list and find the agent id and state from the message and record it in its memory.

```
int MyFunction(xmachine_memory* agent,
xmachine_message_list* list) {
    xmachine_message* message = get_first_message(list);
    while(message) {
        if (message->id == agent->id) {
            agent->state += message->state;
            return 0;
        }
        message = get_next_message(message, list);
    }
    return 0;
}
```

However the above code does not work in FLAME GPU, causing the simulation to crash or hang. Instead an extra flag 'finished' needs to be introduced to tell the code to leave the while loop.

```
int MyFunction(xmachine_memory* agent,
xmachine_message_list* list) {
    bool finished = false;
    xmachine_message* message = get_first_message(list);
    while(message) {
        if (!finished) {
            if (message->id == agent->id) {
                agent->state += message->state;
                finished = true;
            }
        }
        message = get_next_message(message, list);
    }
    return 0;
}
```

This behavior was only observed in the GPU version, raising a concern if while loops have to be explicitly broken for GPU execution when compared to the grid version.

3.1.1.5 Discrete versus continuous

In the GPU version, agents need to be defined in advanced if they are of natures – discrete or continuous agents. This affects how their messages are parsed and handled during the simulation. This required modellers to understand this in advance which was not seen in the grid version. The grid version allows easier writing of the agents and all are handled the same way.

3.1.1.6 Agent birth and death

Both architectures were able to handle agent addition similarly. Similar to the problem of dynamic memory allocation, agents can be introduced in the system if predetermined for the GPU. Every simulation the agent would be introduced using an environment agent, which keeps track of the maximum agents and generate a new agent by using the thread_id :

ID= Maxid +thread ID, where thread ID = blockIDx.x *block Dimx+threadId.x

This should guarantee that all the generated ID's are unique (although they won't be sequential), without using complicated atomic operations. However the GPU could only add 1 agent at a time step, whereas the Grid version could add multiple agents per step.

3.1.1.7 Real time visualization

The Grid version allows simulations to run as batch files producing results on disk. The results have to later downloaded and processed to find data patterns for the simulations. The GPU was much

simpler to do this, as it immediately integrates with a visualization engine. The agent could be observed as the simulation happens and no time waits were needed to visualize the simulations.

4. FUTURE WORK

The current lessons from FLAME are now being ported as a simulation service on the cloud. Table 1 presents why this would be beneficial comparing it to the grid versions.

Table 1. Case for Cloud computing for ABMs

Issues	High performance computing	Cloud computing
Kind of models and processing	The processing is limited to the nature of simple equation models to be processed.	Can introduce dynamic scalability for more complex processing.
Cost	Access to expensive hardware to model and simulate systems.	Resources can be hired as needed.
Failure recovery	No fault recovery when disk space runs out.	Applications can burst to more Clouds if needed, automatically.
Dynamic changes in the model	No real time processing, jobs are submitted to a queue, which means real time changes cannot be incorporated in the models.	Can execute jobs on the fly which can read real time data feeding to the models directly.

In addition to the advantages, the cloud based systems introduces multiple levels of complexity which are discussed in Figure 5.

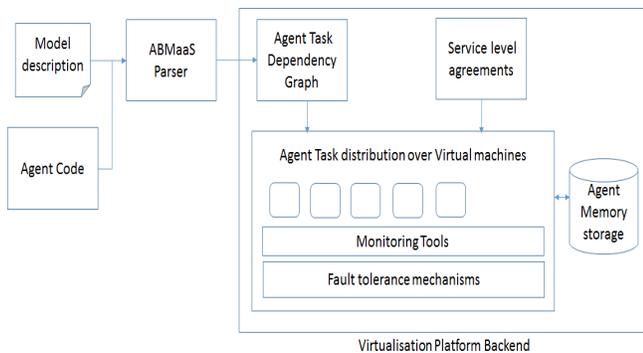


Figure 5. Block diagram of FLAME –Cloud.

Figure 5 describes the architecture of ABMaaS platform. The agent code and model description are input to an ABMaaS parser which is able to perform task distribution and allocation in order to allocate resource efficiently on the backend platforms. The platform providers will have certain service level agreements that need to be consulted when resources are pooled and distributed with the nature of simulations. Depending on the demands, virtual machines can be allocated. These machines are monitored and communicated to a central database to store agent data which is continuously being referenced to during the simulation processing on the machines. Further mechanism such as fault tolerance have to be implemented which will be prevent the high demand of the simulation to affect the active virtual machines.

However the current design faces a multiple additional challenges such as workload balance over virtual machines, model

verification issues, fault tolerance, security and performance optimization. Each of these are still open research problems which need to be solves separately before the system is put together.

5. CONCLUSIONS

Modellers and decision makers are increasingly looking at simulations larger and larger simulations. The motivation is clear – more realistic simulations requires larger populations, or multiple types of population, as the validity of emergent characteristic dependent on both: the accuracy of the behaviour modelled and population sizes. In addition there is a need to forecast behaviours of systems faster than the wall-clock time, and run-time costs are presently inhibiting the effective use of ABM as forecasting tool.

Agent-based models have successfully been able to uncover new aspects of economic systems such as the effect of migration on EU labor markets [1] or uncovering some underlying facts in biological systems [6, 7]. In agent-based modelling, the agents can be complex (e.g. humans), or simple (e.g. ants), with varying memory and functions. Simulating natural systems, researchers have shown how complex ant colony optimizations can be used to help solve complex network routing problems [8] or study the biological transcription functions in cells or bacterial behavior living in human tissues [6, 7].

The current Agent-based Modelling Environments can simulate regional economies on supercomputers – focusing on a number of markets – labour, credit, financial – integrated into a large model running on supercomputers. The frameworks need to be extended to provide a more powerful mechanism for dealing with the complex agent behavior and cloud environments could enable on demand high performance computing to solve some of the current issues faced by present architectures. Even with this route, models written for HPC and GPU should portray similar characteristics, hiding away much of the software complexity from the non computing scientists using the tools to write their models which is a challenge in its own right.

6. ACKNOWLEDGMENTS

Our acknowledgement to the whole FLAME group developers and modellers for allowing us to study both models in detail.

7. REFERENCES

- [1] Grimm, Volker; Railsback, Steven F. (2005). Individual-based Modeling and Ecology. Princeton University Press. p. 485. ISBN 978-0-691-09666-7
- [2] Coakley ST, Holcombe, M. Smallwood R, From molecules to insect communities - how formal agent based computational modelling is uncovering new biological facts, *Scientiae Mathematicae Japonicae Online*, pp.765–778, 2006.
- [3] M. Kiran, P. Richmond, M. Holcombe, L. S. Chin, D. Worth, and C. Greenough, FLAME: Simulating large populations of agents on parallel hardware architectures, in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, 2010, pp. 1633–163
- [4] M. Holcombe, S. Chin, S. Cincotti, M. Raberto, A. Teglio, S. Coakley, C. Deissenberg, S. vander Hoog, C. Greenough, H. Dawid, M. Neugart, S. Gemkow, P. Harting, M. Kiran, and D. Worth, Large-scale modelling of economic systems, *Complex Systems*, no. 2, pp. 175–191, 2012.
- [5] M. Pogson, M. Holcombe, R. Smallwood, and E. Qvarnstrom, Introducing spatial information into predictive NF-kB modelling – an agent-based approach, *PLoS ONE*, vol. 3, no. 6, p. e2367, 2008.

- [6] S. Maleki-Dizaji, M. Holcombe, M. Rolfe, P. Fisher, J. Green, R. Poole, and A. Graham, A systematic approach to understanding Escherichia coli responses to oxygen from microarray raw data to pathways and published abstracts, *Online Journal of Bioinformatics*, vol. 10, no. 1, pp. 51–59, 2011
- [7] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough, Exploitation of high performance computing in the flame agent-based simulation framework, in *2012 IEEE High Performance Computing and Communication*, June 2012, pp. 538–545
- [8] Richmond P., Romano D. (2008), A High Performance Framework For Agent Based Pedestrian Dynamics On GPU Hardware, *Proceedings of EUROSIS ESM 2008 (European Simulation and Modelling)*, October 27-29, 2008, Universite du Havre, Le Havre, France
- [9] M. Burkitt, M. Kiran, S. Konur, M. Gheorghe, F. Ipate Agent-based High-Performance Simulation of Biological Systems on the GPU, *IEEE International Conference on High Performance Computing and Communications*, Aug, 2015, accepted
- [10] Gómez-Sanz J.J., Gervais M.P, Weiss G., 2004, Survey on Agent-Oriented Software Engineering Research, *Methodologies and Software Engineering, Multiagent Systems, Artificial Societies and Simulated Organizations*.
- [11] Wooldridge M., 1998, Agent-based software engineering, *IEEE Proceedings software* 144(1)
- [12] Rao A., Georgeff M., 1993, A model theoretic approach to verification of situated reasoning systems. *Artificial Intelligence*.