Creating GPU-Enabled Agent-Based Simulations Using a PDES Tool

Worawan Marurngsith and Yanyong Mongkolsin

Department of Computer Science, Faculty of Science and Technology, Thammasat University
99 Phaholyothin Road, Pathum Thani, 12120, Thailand
wdc@cs.tu.ac.th, 5109035054@student.cs.tu.ac.th

Abstract. By offloading some computation to graphical processing units (GPUs), agent-based simulation (ABS) can be accelerated up to thousands of times faster. To exploit the power of GPUs, modellers can use available simulation frameworks to auto-generated GPU codes without requiring any knowledge of GPU programming languages. However, such frameworks only support computation on the GPUs of a particular vendor. This paper proposes techniques, implemented in a synchronous parallel discrete event simulation (PDES) tool, to allow modellers to create ABS models, and to specify computation regions in the models for multiple vendor's GPUs or CPUs. The technique comprises a set of meta-language tags and a compilation framework to convert user-defined GPU execution regions to OpenCL. A well-known cellular ABS models, the Conway's Game of Life, have been implemented and evaluated on two platforms *i.e.*, the NVIDIA GeForce 240M LE and AMD Radeon HD6650M. The preliminary results demonstrate two findings: (a) the proposed technique allows the example ABS model to be executed on a PDES engine successfully; (b) the generated GPU-enabled ABS model can achieve fourteen times faster than its multicore version.

Keywords. OpenCL, GPU, Agent-based simulation, PDES, Acceleration.

1 Introduction

Exploiting the computational power of graphic processing units (GPUs), high performance agent-based simulations (ABS) permit modellers to study large-scale complex phenomena faster than before. Development of languages and tools has made the integration of GPU computation to ABS models become more user friendly. Among several GPU languages[1] (*e.g.*, Brook, CUDA, and OpenCL), CUDA is probably the most popular language used in ABS as they are several parallelisation techniques and optimisation available[2, 3]. Despite its popularity,

CUDA-based ABS models suffer from lack of portability as they are limited to executing only on NVIDIA GPUs. To overcome this limitation, recent GPU languages including, OpenCL[4] and C++AMP[5], can be a good candidate. Both languages allow programmers to express computation on heterogeneous processing platforms including CPUs and GPUs (and any compliant devices). The

S. Omatu et al. (Eds.): *Distrib. Computing & Artificial Intelligence,* AISC 217, pp. 227–234.
DOI: 10.1007/978-3-319-00551-5_28

OpenCL language has been used in Multi2Sim [6] framework to accelerate parallel discrete event simulations (PDES). However to the best of the authors' knowledge, there are still no ABS frameworks based on OpenCL or C++ AMP.

As significant performance improvements are necessary, the implementation of many legacy ABS simulation frameworks *e.g.*, EcoLab, NetLogo, Mason, Repast (HPC version), ABM++ and FLAME have shifted towards parallel performance and scalability using multi-threading and multiprocessing techniques [7]. Only two ABS frameworks *i.e.*, the work of Lysenko and D'Souza [8] and FLAME[9], provide performance acceleration of ABS models using GPUs.

The first framework [8], developed at the Michigan Technological University, is a completely GPU-based ABS framework using C++, OpenGL and GLSL. The framework composed of three parts *i.e.*, a system for handling environments, mobility mechanism in agents, and visualisation. Mobile agents are off-loading to GPUs and encoding into textures. Some algorithms have been employed to perform three basic tasks of mobile agents on GPUs, including: storing and updating the mobile agent state and connecting the mobile agents to their environment. The experimental results show that using the framework, an ABS model can achieve a speed up factor of over 9,000.

FLAME [9] is the flexible large-scale agent modelling environment developed at the University of Sheffield. The framework is template-based, ideally suited for creating ABS models of cellular systems running on multicore and GPUs. Simulations are advanced by using discrete time steps. Modellers use the FLAME's template, based on XML syntax, to describe all GPU computations without having to have any specialist knowledge of the underlining GPU platform. The template is then automatically translated into the corresponding CUDA codes. Using the auto-generated GPU codes in FLAME, simulation of cellular models could achieve hundreds of times faster than they multicore version. Model data can also be saved and analysed post simulation; or viewing on the fly. Published experimental results have shown that, using FLAME, cellular models can be massively accelerated using the parallel GPU architecture beyond that of CPU-based frameworks.

Existing simulation frameworks have confirmed that GPU computing can be a novel cost-effective approach towards high-performance ABS models, with less programming effort from modellers. Some research has shown the successful parallelism techniques for achieving high-performance ABS models on multicores and GPUs *i.e.*, cellular automata models using CUDA with OpenMP[3]. adaptive Swarm model using Java-binding OpenCL [10]. However, developing a simulation framework for ABS models to exploit computational power of multicores and GPUs at the same time is still a challenge.

This paper presents the use of a synchronous parallel discrete-event simulation (PDES) tool, called P-HASE[1], as a framework for creating GPU-based ABS models using OpenCL C++ binding. Our main contributions are the following:

[1] Available at `http://parlab.cs.tu.ac.th/P-Hase`

- The mapping of ABS components to the discrete event simulation (DES) components is presented,
- A technique to convert user-defined GPU computation regions to OpenCL kernels is proposed,
- An example GPU-based ABS model is presented; and the performance analysis experiment of the model on two GPUs models has shown the speedup up to fourteen-fold, in comparison to its multicore counterpart.

The rest of the paper is organised as follow. The next section presents the techniques used to create ABS models and to integrate them with GPU computation. Section 3 presents the detail of compilation framework for generating OpenCL codes. An example ABS model and the experimental results are presented in Section 4. Section 5 gives the conclusion of the paper.

2 GPU-Enabled ABS on P-HASE Simulation Tool

P-HASE [11] is a parallel extension of HASE [12] (the Hierarchical computer Architecture design and Simulation Environment), which is a framework for discrete-event simulation (DES) and visualisation, developed at the University of Edinburgh. The key idea of HASE is two folds. First it aims to provide modellers facilities to allow a simulation model to be hierarchically structured. Second it provides modellers the visual verification of a model's behaviour via an animation. The P-HASE simulation tool derives all functionalities from HASE. Thus, it supports the creation, execution of simulation models, and post-mortem debugging via an animation. Extra to HASE is that P-HASE provides a scalable PDES engine using conservative synchronous technique [11]. The tool generates a PDES from the HASE DES definition. Thus, sequential models can be accelerating by the parallel engine provided in P-HASE.

Table 1 Mapping between components of P-HASE model to the ABS model

DES Components in P-HASE	ABS Components
Entity	Environment, static agent
Ports	Interaction between environments/static agents
Task object inside an Entity	Agents
Compound entity	Hierarchical environments
Design template	Multiple environments connected via a topology

There are several ways to represent ABS models in a specification which conform with the discrete event system specification (DEVS) formalism **[13]**. As diagram shown in Fig.1, to create a simulation model in P-HASE, modellers first have to provide a structural definition of model's components using a file written in the entity description language (EDL). Components of a model in P-HASE are atomic units called *Entities* that communicate by passing events via *Ports*. Each

Entity can be viewed as an independent thread. Since it is better to group agents which frequently communicate with each other on one thread, the Entity is used to represent an Environment (or virtual space) in ABS. So that Ports can be used to represent the communication between Environments.

Modellers can group together sets of Entities that are logically related in two ways: (a) aggregating them into a *compound entity*, or (b) using a *design template*. Compound entities describe the vertical composition of a more complex entity from its basic subunits up to its higher abstract-level units. A design template describes functional relationships among entities and also provides the means of connection between them. Thus, a group of Environments in ABS can be represented in three ways: (a) as multiple Entities connected via ports, (b) as a compound entity or (c) as a design template. Table 1 shows the possible mapping between DES components used in P-HASE to the ABS components.

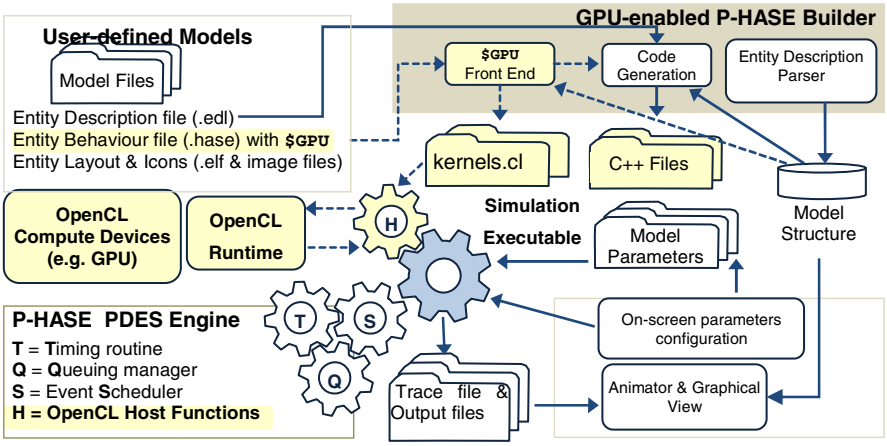

Fig. 1 Modules of the P-HASE tool with GPU-enabled extension

The top left corner of Fig.1 depicts all types of input file needed for creating a model. The key file type for adding the GPU computation to a model is the behaviour file (*.hase*). Once the structure of model components (*.edl file*) has been defined, modellers insert C++ codes to mimic the behaviour of each entity in *.hase* file. The behaviour file is a text file organised in sections separated by tags (e.g., **`$startup`**, **`$pre`**, **`$phase_0`**, **`$phase_1`**, **`$report`**). Modellers write C++ code into each section to implement the behaviour for each particular time step. For example, in synchronous models using two-phase clocking, the behaviour of each clock phase must be added under the **`$phase_0`** and **`$phase_1`** sections respectively. Codes specified under these sections will be generated by the HASE Builder into a clock-phase routine. In a clock phase routine, a special tag (**`$GPU`**) is used to specified the beginning of a GPU computation. The tag must be immediately followed by a **`for`** loop which implements the behaviour of agents. As show in Fig.1, when modellers compile the model, the behaviour file with **`$GPU`** tag is passed to the **`$GPU`** Front End module. The tag instructs the module to generate

loop iterations as agents. The module performs some semantic analysis on the following **`for`** loop and generates the corresponding OpenCL codes. The generated codes have two parts. The first part is generated as a file with *.cl* extension, called *kernels*. It is a file containing C-style functions each of which describes a unit of task to be offloaded to GPUs (so called *devices*). The body of the **`for`** loop is generated as a kernel function. All array references are flattening into one dimension to fit with the OpenCL device architecture. The second part, called *host*, is source codes used to initialise the OpenCL execution environment (*Context*), and to build and compile the kernel. The host codes are generated and inserted into the behaviour file to replace the **`for`** loop.

The modified behaviour files are passed to the original code generation process in HASE to build the simulation executable. During a simulation run, the simulation executable acts as the host of the OpenCL environment. Thus, it will create a context and launch kernels to GPUs.

3 Compilation Framework and the GPU-Enable PDES

Fig. 2 depicts the structure of the compilation framework implemented in P-HASE for translating each **`$GPU`** tags to an OpenCL kernel, and inserting the calls from host. Similar techniques used in our previous work have been adapted to perform static analysis and code generation [14]. Fig.3 shows the mechanism of GPU-enabled PDES. During a simulation run, each Entity represents an Environment where agents reside. Each Environment is created as a thread executing on CPUs, deriving from a clock abstract class for timing control. Events are used for time advancing. At the start of a simulation, the Environment initialises the OpenCL host functions. These includes querying for platforms (CPUs or GPUs) or *compute devices*, and creating *Context* on the devices found; creating command queue in the context; reading kernels.cl into a stream and creating a program object from it; specifying a kernel function from the program object and creating it as a kernel object; creating memory buffers for passing inputs and receiving results. At each clock phase routine, Environment starts agents by submitting the kernel object to the command queue for executing on GPUs. Each kernel object will be created as an execution unit, called *work-item*, and assigned to each ALU of the compute device. When agents finished the tasks of each time step, results are written back to Environment via buffers.

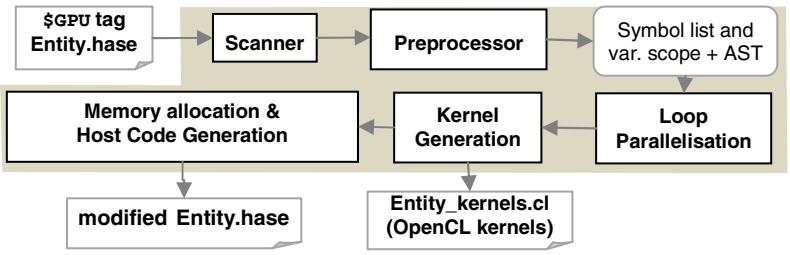

Fig. 2 Overview of the compiler framework implemented in the P-HASE Builder

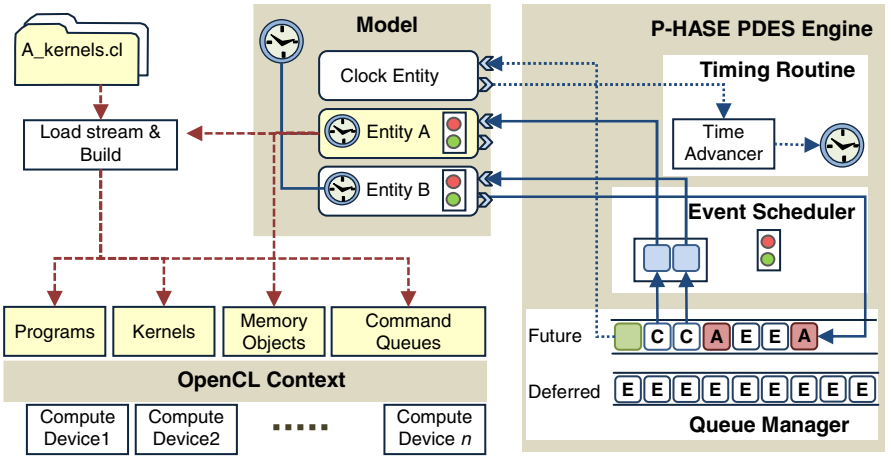

Fig. 3 Structure and mechanism of the P-HASE Tool (GPU related modules are highlighted)

4 Preliminary Experiments and Results

Our proposed platform is still limited to cellular models which tasks of agents are written in a loop. The Conway's Game of Life (GOL) with two-dimensional grid has been developed in the P-HASE tool (Fig.4) in two versions, the model with 1- and 2- environments (GOL-1, and GOL-2). The environment represents the $n \times m$ grid of cells. In the environment, a cell lives in each grid entry and cannot move. Each version receives the number of agents as a parameter; and has been validated by matching the model results against the CPU-only models.

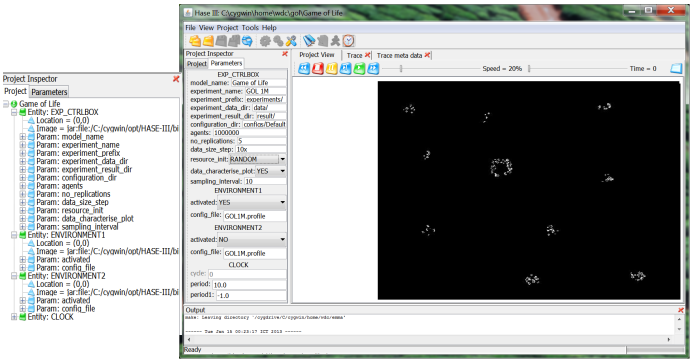

Fig. 4 The ABS model of Conway's Game of Life on P-HASE

Preliminary experiments have been done using two middle class graphics card for laptops, the AMD Radeon HD6650M and the NVIDIA GeForce 240LE. The setup of the experiments aimed for finding the speedup achieved on using

GPU-enabled models generated by P-HASE if modellers are mobile. That is if the simulation is run on-site where any high performance computation platform is not applicable. The experiments used the GOL-1 model (Fig.5) and GOL-2 model (Fig.6) with four different numbers agents. Each model was repeated five times, and executed on a core-2 duo CPU, a Core i7-2620M CPU, and on the two graphics cards. Execution time was recorded for each run and the speedup was calculated by using the dual-core execution time as a base line.

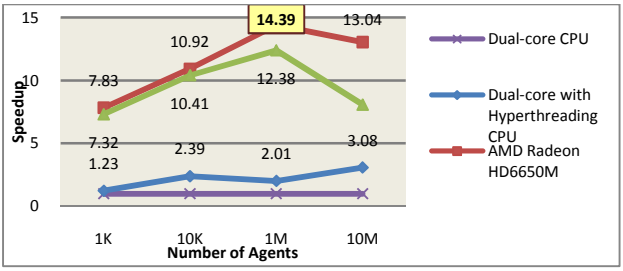

Fig. 5 Speedup of three platforms compared to dual-core model for single environment model

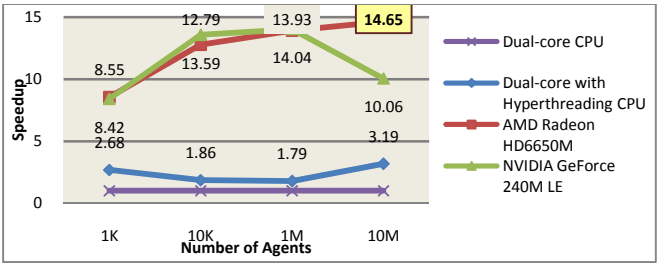

Fig. 6 Speedup of three platforms compared to dual-core model for two environments model

Overall results show that GPU-enabled models outperform the dual-core versions for 7 – 14 times, and outperform the Core i7 model for 5 – 12 times. The speedup plots of GPU-enabled models drop when the agent size is 10,000,000, which may cause by the synchronisation overhead. Detail analysis has to be done to quantify the overhead and to improve the code generation process.

5 Conclusion

The use of a synchronous PDES tool, called P-HASE, as a framework for creating a GPU-based ABS models using OpenCL C++ has been presented. A cellular ABS model was evaluated and confirmed the feasibility to use the PDES framework to create efficient GPU-based models. The work presented in this paper is a

preliminary wor. The framework is still limited to represent cellular models with simple behaviours. Thus our immediate future work is to explore the ways to cover more realistic models to understand the performance limitation.

Acknowledgement. We wish to thank Professor Roland Ibbett and David Dolman for allowing us to extend the original HASE tool. We thank the referees for valuable comments.

References

1. Brodtkorb, A.R., Hagen, T.R., Sætra, M.L.: Graphics processing unit (GPU) programming strategies and trends in GPU computing. Journal of Parallel and Distributed Computing 73(1), 4–13 (2013)
2. Perumalla, K.S., Aaby, B.G.: Data parallel execution challenges and runtime performance of agent simulations on GPUs. In: Proceedings of the SpringSim. SCS, pp. 116–123 (2008)
3. Falk, M., et al.: Parallelized agent-based simulation on CPU and graphics hardware for spatial and stochastic models in biology (2011)
4. Group, K.: OpenCL - The open standard for parallel programming of heterogeneous systems (2013), http://www.khronos.org
5. Microsoft. C++ AMP (C++ Accelerated Massive Parallelism) (2013), http://msdn.microsoft.com/en-us/library/vstudio/hh265137.aspx
6. Ubal, R., et al.: Multi2Sim: a simulation framework for CPU-GPU computing. In: Proceedings of PACT 2012, pp. 335–344. ACM, Minneapolis (2012)
7. Coakley, S., et al.: Exploitation of High Performance Computing in the FLAME Agent-Based Simulation Framework. In: 2012 IEEE HPCC-ICESS (2012)
8. Lysenko, M., D'Souza, R.M.: A Framework for Megascale Agent Based Model Simulations on Graphics Processing Units. J. of Art. Soc. and Soc. Sim. 11(4), 10 (2008)
9. Richmond, P., et al.: High performance cellular level agent-based simulation with FLAME for the GPU. Briefings in Bioinformatics 11(3), 334–347 (2010)
10. Laville, G., et al.: Using GPU for multi-agent multi-scale simulations, pp. 197–204 (2012)
11. Mongkolsin, Y., Marurngsith, W.: P-HASE: An Efficient Synchronous PDES Tool for Creating Scalable Simulations. In: Xiao, T., Zhang, L., Fei, M. (eds.) AsiaSim 2012, Part III. CCIS, vol. 325, pp. 231–245. Springer, Heidelberg (2012)
12. Coe, P.S., et al.: Technical note: a hierarchical computer architecture design and simulation environment. ACM Trans. Model. Comput. Simul. 8(4), 431–446 (1998)
13. Tauböck, S., et al.: The <morespace> Project: Modelling and Simulation of Room Management and Schedule Planning at Universityby Combining DEVS and Agent-based Approaches. J. on Developments and Trends in Mod. and Simulation 22(2), 11–20 (2012)
14. Makpaisit, P., Marurngsith, W.: Griffon - GPU programming APIs for scientific and general purpose computing (Extended version). International Journal of Artificial Intelligence 8(12 S), 223–238 (2012)